

Coloring tournaments with forbidden substructures

Krzysztof Choromanski
Google Research
New York, NY, USA

Tony Jebara
Columbia University
New York, NY, USA

July 28, 2012; revised April 7, 2015

Abstract

Coloring graphs is an important algorithmic problem in combinatorics with many applications in computer science. In this paper we study coloring tournaments. A chromatic number of a random tournament is of order $\Omega(\frac{n}{\log(n)})$. The question arises whether the chromatic number can be proven to be smaller for more structured nontrivial classes of tournaments. We analyze the class of tournaments defined by a forbidden subtournament H . This paper gives a first quasi-polynomial algorithm running in time $e^{O(\log(n)^2)}$ that constructs colorings of H -free tournaments using only $O(n^{1-\epsilon(H)} \log(n))$ colors, where $\epsilon(H) \geq 2^{-2^{50|H|^2+1}}$ for many forbidden tournaments H . To the best of our knowledge all previously known related results required at least sub-exponential time and relied on the regularity lemma. Since we do not use the regularity lemma, we obtain the first known lower bounds on $\epsilon(H)$ that can be given by a closed-form expression. As a corollary, we give a constructive proof of the celebrated open Erdős-Hajnal conjecture with explicitly given lower bounds on the EH coefficients for all classes of prime tournaments for which the conjecture is known. Such a constructive proof was not known before. Thus we significantly reduce the gap between best lower and upper bounds on the EH coefficients from the conjecture for all known prime tournaments that satisfy it. We also briefly explain how our methods may be used for coloring H -free tournaments under the following conditions: H is any tournament with ≤ 5 vertices or: H is any but one tournament of six vertices.

Keywords: coloring tournaments, the Erdős-Hajnal conjecture, transitive subtournaments

1 Introduction

Let $|\cdot|$ to denote the size of the set. Let G be a graph. We denote by $V(G)$ the set of its vertices and by $E(G)$ the set of its edges. Sometimes instead of writing $|V(G)|$ we use the shorter notation $|G|$. We call $|G|$ the *size of G* . For a subset $S \subseteq V(G)$ we denote by $G|S$ the subgraph of G induced by S . A *clique* in an undirected graph is a set of pairwise adjacent vertices. An *independent set* in the undirected graph is a set of pairwise nonadjacent vertices. All logarithms used in this paper are of base 2.

A *tournament* is a directed graph such that, for every pair v and w of vertices, exactly one of the edges (v, w) or (w, v) exists. If (v, w) is an edge in the tournament then we say that v is *adjacent to w* and w is *adjacent from v* . The *indegree* of a vertex v of a tournament T is the number of vertices $w \in V(T)$ such that $(w, v) \in E(T)$. Similarly, the *outdegree* of a vertex v of a tournament T is the number of vertices $w \in V(T)$ such that $(v, w) \in E(T)$. A *directed cycle* is a set of vertices $\{v_0, \dots, v_{k-1}\}$ for some $k \geq 3$ such that $(v_i, v_{(i+1) \bmod k})$ is a directed edge for $i = 0, \dots, k-1$. A tournament is *transitive* if it contains no directed cycle. For the set of vertices $V = \{v_1, v_2, \dots, v_k\}$ we say that an ordering (v_1, v_2, \dots, v_k) is *transitive* if v_1 is adjacent to all other vertices of V , v_2 is adjacent to all other vertices of V but v_1 , etc. A subset $S \subseteq V(T)$ is *transitive* if it induces a transitive tournament. For a tournament H we say that a tournament T is H -free if T does not contain H as an induced subtournament.

A *proper coloring* of a tournament T is an assignment of colors to its vertices such that there is no directed monochromatic cycle. Equivalently, we can consider a proper coloring of a hypergraph, where the set of vertices is $V(T)$ and the set of hyperedges consists of all triples of vertices inducing directed triangles in T . When properly coloring a hypergraph, we do not want to create monochromatic hyperedges. The latter is equivalent to our previous definition of coloring since one can easily note that if a monochromatic directed cycle exists in T then a monochromatic directed triangle exists as well. The *chromatic number* of a tournament T is the minimal number of colors needed to properly color T . Note that under every proper coloring of the vertices of T each color class induces a transitive subtournament. Let T_n^r be a random n -vertex tournament, where independently for every pair of vertices $\{u, v\}$, we have: $(u, v) \in E(T)$ with probability $\frac{1}{2}$. It is not hard to prove that with probability tending to 1 as $n \rightarrow \infty$ the largest transitive subtournaments of T_n^r are of logarithmic size. Thus, according to the remark above, the chromatic number of a random tournament is $\Omega(\frac{n}{\log(n)})$.

A celebrated unresolved conjecture of Erdős and Hajnal states that:

Conjecture 1.1 *For every tournament H there exists $\epsilon(H) > 0$ such that every n -vertex H -free tournament contains a transitive subtournament of size at least $n^{\epsilon(H)}$.*

In fact the conjecture was first proposed in the undirected setting by Erdős and Hajnal but was proven to be equivalent to the directed setting above by Alon, Pach and Solymosi in 2001 (see: [1]). The undirected version of the conjecture (see: [2]) states that:

Conjecture 1.2 *For every undirected graph H there exists $\epsilon(H) > 0$ such that every n -vertex graph G that does not contain H as an induced subgraph contains a clique or an independent set of size at least $n^{\epsilon(H)}$.*

If for a given tournament H there exists $\epsilon(H) > 0$ then we say that H *satisfies the Erdős-Hajnal conjecture with $\epsilon(H)$* or simply: H *satisfies the Erdős-Hajnal conjecture*. The coefficient $\epsilon(H)$ in

the statement is called the *EH coefficient*. From now on instead of saying: "the conjecture of Erdős and Hajnal" we will simply say: "the conjecture". From the context it will be always clear whether we have in mind a directed or an undirected version.

A subset of vertices $S \subseteq V(H)$ of a tournament H is called *homogeneous* if for every $v \in V(H) \setminus S$ the following holds: either $\forall_{w \in S}(w, v) \in E(H)$ or $\forall_{w \in S}(v, w) \in E(H)$. A homogeneous set S is called *nontrivial* if $|S| > 1$ and $S \neq V(H)$. A tournament is called *prime* if it does not have nontrivial homogeneous sets.

We call by C_5 a unique tournament on five vertices, where each vertex has indegree and outdegree two.

2 Main results and related work

Now we summarize our main results. We present a family of tournaments called *constellations* and show a quasi-polynomial algorithm running in time $e^{(\log(n))^2}$ that constructs a proper coloring of an H -free tournament T , where H is a constellation, with $O(n^{1-\epsilon(H)} \log(n))$ colors, where $\epsilon(H) = 2^{-2^{50|H|^2+1}}$. We prove that:

2.1 *If H is a constellation then every n -vertex H -free tournament may be properly colored with $n^{1-\epsilon(H)} \log(n)$ colors, where $\epsilon(H) = \frac{1}{2^{250h^2+1}}$ and $h = |H|$. Furthermore, every H -free tournament T contains a transitive subtournament of order at least $|T|^{\epsilon(H)}$.*

Constellations play important role in the conjecture since all known prime tournaments with more than six vertices satisfying the conjecture are constellations. Furthermore all tournaments for which the conjecture has been proven so far can be obtained from an infinite family of constellations and three other tournaments after applying the so-called substitution procedure introduced in [1]. Prime tournaments are important since if the conjecture is true for prime tournaments then it is true in general. All tournament apart from a tournament C_5 and some two six-vertex tournaments, considered in papers such as: [5], [7], [6], are special examples of tournaments that can be obtained from constellations using the substitution procedure. We will briefly explain how, as a byproduct of our techniques, one can color H -free tournaments with $O(n^{1-\epsilon})$ colors for every tournament H on at most 5 vertices and all but one tournament H on 6 vertices. Our techniques give a constructive proof of the Erdős-Hajnal conjecture for all the constellations. Besides, after combining our methods with the substitution procedure, we obtain explicit lower bounds on EH coefficients for all known tournaments satisfying the conjecture. The family of constellations was introduced in [4]. In the same paper the conjecture was proven for them. However no algorithm to construct the $O(n^{1-\epsilon})$ -coloring efficiently was given. Furthermore, even though that paper showed that an optimal coloring uses $O(n^{1-\epsilon})$ colors, the constant $\epsilon > 0$ was extremely small since the proof heavily relied on the regularity lemma. In this paper we show that the regularity lemma is not needed at all and thus we obtain much better bounds. Our main contribution is an algorithmic proof that does not use the regularity lemma and a general method that can be used to show first explicit lower bounds on EH coefficients for all known tournaments satisfying the conjecture. This leads to better understanding of the asymptotics of EH coefficients. All previously known positive results regarding the conjecture for prime tournaments needed the regularity lemma. That implied big gaps between best known upper bounds on $\epsilon(H)$ of the order $\frac{1}{|H|}$ and best known lower bounds that were inversely proportional to the Szemerédi tower function. In this paper we significantly reduce this gap. In their original paper Erdős and Hajnal asked how the EH coefficient depends on

graph parameters such as its size. Our paper provides a step towards an answer for prime graphs. Previous results concerning upper bounds for EH coefficients of prime graphs were given (see: [3], [4]) but these lacked explicit lower bounds. This paper attempts to fill the gap.

Our techniques may be easily adapted to other problems regarding coloring classes of tournaments defined by forbidden subtournaments.

Let us end this section by briefly summarizing recent progress regarding the directed version of the conjecture. [6] described all tournaments satisfying the conjecture in the strongest, linear sense. Similarly, all tournaments satisfying the conjecture in an almost linear sense (so-called *pseudocelebrities*) were described by [7]. However, both results are for tournaments that are not prime. In [5] and [4] several results regarding the conjecture for prime tournaments were proven. All the previous positive results were of purely theoretical flavor and did not easily translate into algorithmic results.

The paper is organized as follows:

- In Section 3 we formally define the family of constellations and introduce other important definitions.
- In Section 4 we give an algorithm to color H -free tournaments for all constellations H .
- In Section 5 we prove correctness and analyze the running time of the presented algorithm and therefore prove Theorem 2.1.
- In Section 6 we discuss some further applications of the introduced techniques.

3 Constellations

In this section we define the family of constellations and introduce other important definitions that will be used in further analysis.

Fix some ordering of vertices of a tournament H . An edge (v, w) under this ordering is called a *backward edge* if w precedes v in this ordering. Let T be a tournament with vertex set $V(T)$ and fix some ordering of its vertices. The *graph of backward edges* under this ordering, denoted by $B(T, \theta)$, has vertex set $V(T)$, and $\{v_i, v_j\} \in E(B(T, \theta))$ if and only if (v_i, v_j) or (v_j, v_i) is a backward edge of T under the ordering θ . For an integer t , we call the graph $K_{1,t}$ a *star*. Let S be a star with vertex set $\{c, l_1, \dots, l_t\}$, where c is adjacent to vertices l_1, \dots, l_t . We call c the *center of the star*, and l_1, \dots, l_t the *leaves of the star*. Note that in the case $t = 1$ we may choose arbitrarily any one of the two vertices to be the center of the star, and the other vertex is then considered to be the leaf. Let $\theta = (v_1, v_2, \dots, v_n)$ be an ordering of the vertex set $V(T)$ of a n -vertex tournament T . For a subset $S \subseteq V(T)$ we say that $v_i \in S$ is a *left point of S under θ* if $i = \min\{j : v_j \in S\}$. We say that $v_i \in S$ is a *right point of S under θ* if $i = \max\{j : v_j \in S\}$. If from the context it is clear which ordering is taken we simply say: *left point of S* or *right point of S* . For an ordering θ and two vertices v_i, v_j with $i \neq j$ we say that v_i is *before* v_j if $i < j$ and *after* v_j otherwise. We say that a vertex v_j is *between* two vertices v_i, v_k under an ordering $\theta = (v_1, \dots, v_n)$ if $i < j < k$ or $k < j < i$.

A *right star* in $B(T, \theta)$ is an induced subgraph with vertex set $\{v_{i_0}, \dots, v_{i_t}\}$, such that $B(T, \theta)|\{v_{i_0}, \dots, v_{i_t}\}$ is a star with center v_{i_t} , and $i_t > i_0, \dots, i_{t-1}$. In this case we also say that $\{v_{i_0}, \dots, v_{i_t}\}$ is a right star in T . A *left star* in $B(T, \theta)$ is an induced subgraph with vertex set $\{v_{i_0}, \dots, v_{i_t}\}$, such that $B(T, \theta)|\{v_{i_0}, \dots, v_{i_t}\}$ is a star with center v_{i_0} , and $i_0 < i_1, \dots, i_t$. In this case we also say that $\{v_{i_0}, \dots, v_{i_t}\}$ is a left star in T . From now on whenever we will refer to the star in $B(T, \theta)$ we will mean a left star or a right star.

Let H be a tournament and assume there is an ordering θ of its vertices such that every connected component of $B(H, \theta)$ is either a star or a singleton under this ordering. We call this ordering a *star ordering*. The *interstellar graph* of H under a star ordering θ is an undirected graph, whose vertices are the sets of leaves of the stars of H under θ and any two given vertices L_1 and L_2 are adjacent iff:

- the left point of L_1 precedes the right point of L_2 in θ and
- the left point of L_2 precedes the right point of L_1 in θ .

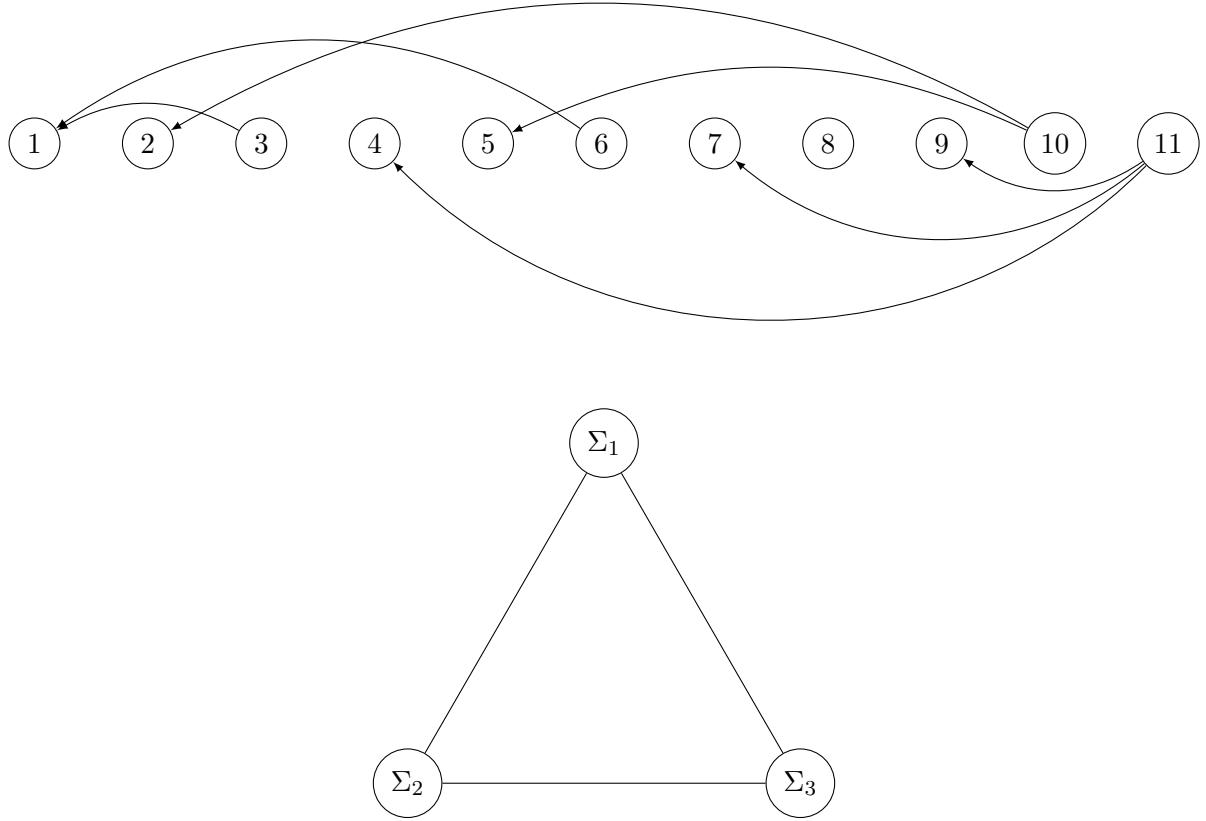


Fig.1 Constellation with backward edges drawn under its constellation ordering and consisting of three stars: $\Sigma_1, \Sigma_2, \Sigma_3$ (above). The interstellar graph where nodes correspond to the stars and two edges are adjacent if and only if the intervals defined by the sets of leaves under given constellation ordering intersect (below).

For each connected component C of the interstellar graph of H denote by $Z(C)$ the union of subsets of $V(H)$ corresponding to its vertices (this is the union of some subsets of $V(H)$ of the vertices of H). Next let us define $\mathcal{C}(Z(C))$ as follows. We say that a vertex $v \in \mathcal{C}(Z(C))$ if $v \in Z(C)$ or v is between some two vertices of $Z(C)$ under the ordering θ . Let C_1, \dots, C_k be the connected components of the interstellar graph. Note that for any given $1 \leq i < j \leq k$ either every vertex of $\mathcal{C}(Z(C_i))$ is before every vertex of $\mathcal{C}(Z(C_j))$, or every vertex of $\mathcal{C}(Z(C_j))$ is before every vertex of $\mathcal{C}(Z(C_i))$. Thus there is a natural ordering of the sets $\mathcal{C}(Z(C_i))$ for $i = 1, 2, \dots, k$ induced by the ordering of the vertices. Denote the ordered sequence of the sets $\mathcal{C}(Z(C_i))$ for $i = 1, 2, \dots, k$ as $(\mathcal{W}_1, \dots, \mathcal{W}_k)$, where a set \mathcal{W}_i is before a set \mathcal{W}_j for $1 \leq i < j \leq k$. Denote $\mathcal{W}_0 = \mathcal{W}_{k+1} = \emptyset$. For

$i = 1, 2, \dots, k + 1$ denote by \mathcal{R}_i the set of the vertices of H that are after all the vertices of \mathcal{W}_{i-1} and before all the vertices of \mathcal{W}_i under the ordering θ . Note that if \mathcal{R}_i is nonempty then all its elements are centers of the stars of H . Denote the set of nonempty sets \mathcal{R}_i as $\{\mathcal{M}_1, \dots, \mathcal{M}_r\}$ for some $r \geq 0$. Note that $\{\mathcal{W}_1, \dots, \mathcal{W}_k, \mathcal{M}_1, \dots, \mathcal{M}_r\}$ is a partition of the vertices of H . Denote this partition by $P_\theta(H)$. We are ready to define constellations.

A tournament T is a *constellation* if there exists a star ordering θ of its vertices such that if a center of a star belongs to some set $P \in P_\theta(H)$ then no leaf of this star belongs to P .

We call such an ordering a *constellation ordering* of T . Let $\Sigma_1, \dots, \Sigma_l$ be the non-singleton components of $B(T, \theta)$. We say that $\Sigma_1, \dots, \Sigma_l$ are the *stars of T under θ* . If $V(T) = \bigcup_{i=1}^l V(\Sigma_i)$, we say that T is a *regular constellation*.

Even though the definition of the family of constellations that we have just presented seems to be complicated, it is in fact easy to construct examples of constellations of an arbitrary size. That is because our definition uses the notion of the constellation ordering and this term has natural and straightforward pictorial interpretation.

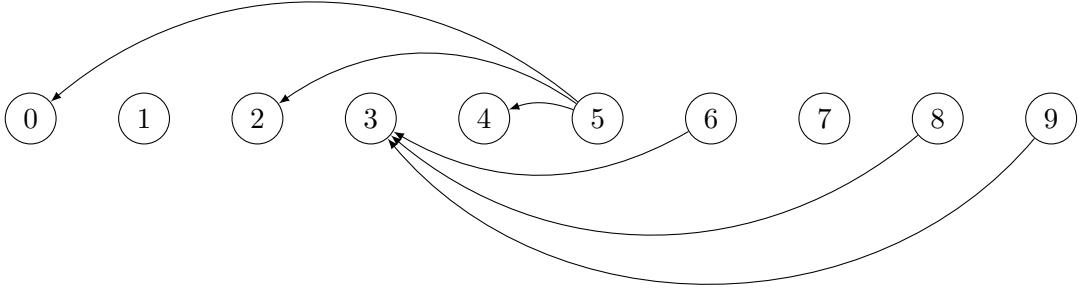


Fig.2 Constellation consisting of two stars - one left and one right. For the clarity of the picture only backward edges were drawn.

A *galaxy ordering* of the vertices of the tournament is the constellation ordering under which no center of the star appears between leaves of another star. Notice that this is not necessarily the case for the constellation ordering. A center of the star can be between leaves (call this set \mathcal{L}) of another star, but if it happens then all its leaves have to be in a different connected component of the interstellar graph than the one that corresponds to \mathcal{L} . A *galaxy* is a tournament that has a galaxy ordering of vertices. Galaxies is a subfamily of constellations. The conjecture was proven for them in [5]. That was the first result where the conjecture was proven for an infinite family of prime tournaments. Our constructive proof gives much better lower bounds on EH coefficients for galaxies than those from [5].

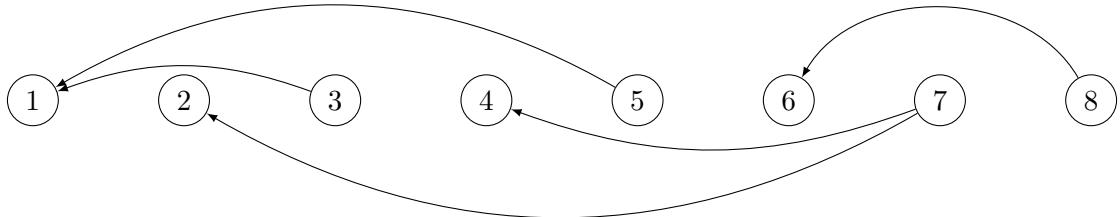


Fig.3 Galaxy consisting of one left and two right stars. All edges that are not drawn are forward.

3.1 Some useful definitions

This section provides several definitions used in the paper.

Take a tournament T . Let $X, Y \subseteq V(T)$ be disjoint, where $|X|, |Y| > 0$. Denote by $e_{X,Y}$ the number of directed edges (x, y) , where $x \in X$ and $y \in Y$. The *directed density from X to Y* is defined as $d(X, Y) = \frac{e_{X,Y}}{|X||Y|}$.

We say that a tournament T is ϵ -transitive if it contains a transitive subtournament of order at least $|T|^\epsilon$.

For the transitive pairwise disjoint subsets $T_1, T_2, \dots, T_k \subseteq V(T)$ we say that a sequence (T_1, T_2, \dots, T_k) is a (c, λ, ϵ) -t-sequence of length k (where t stands for: *transitive*) if the following holds:

- $d(T_i, T_j) \geq 1 - \lambda$ for $1 \leq i < j \leq k$, and
- $|T_i| \geq c|T|^\epsilon$ for $i = 1, 2, \dots, k$.

For the transitive pairwise disjoint subsets $T_1, T_2, \dots, T_k \subseteq V(T)$ we say that a sequence (T_1, T_2, \dots, T_k) is a smooth (c, λ, ϵ) -t-sequence of length k if the following holds:

- $d(\{v\}, T_j) \geq 1 - \lambda$ for $1 \leq i < j \leq k$ and $v \in T_i$,
- $d(T_i, \{v\}) \geq 1 - \lambda$ for $1 \leq i < j \leq k$ and $v \in T_j$, and
- $|T_i| \geq c|T|^\epsilon$ for $i = 1, 2, \dots, k$.

Note that every smooth (c, λ, ϵ) -t-sequence is a (c, λ, ϵ) -t-sequence.

For the pairwise disjoint subsets $A_1, \dots, A_k \subseteq V(T)$ we say that a sequence (A_1, A_2, \dots, A_k) is a (c, λ) -l-sequence of length k (where l stands for: *linear*) if the following holds:

- $d(A_i, A_j) \geq 1 - \lambda$ for $1 \leq i < j \leq k$, and
- $|A_i| \geq c|T|$ for $i = 1, 2, \dots, k$.

For the pairwise disjoint subsets $A_1, \dots, A_k \subseteq V(T)$ we say that a sequence (A_1, A_2, \dots, A_k) is a smooth (c, λ) -l-sequence of length k if the following holds:

- $d(\{v\}, A_j) \geq 1 - \lambda$ for $1 \leq i < j \leq k$ and $v \in A_i$,
- $d(A_i, \{v\}) \geq 1 - \lambda$ for $1 \leq i < j \leq k$ and $v \in A_j$, and
- $|A_i| \geq c|T|$ for $i = 1, 2, \dots, k$.

Note that every smooth (c, λ) -l-sequence is a (c, λ) -l-sequence.

For the pairwise disjoint subsets $A_1, \dots, A_{k+1}, T_1, \dots, T_k \subseteq V(T)$, where T_1, \dots, T_k are transitive, we say that a sequence $(A_1, T_1, \dots, A_k, T_k, A_{k+1})$ is a (c, λ, ϵ) -m-sequence of length $2k + 1$ (where m stands for: *mixed*) if the following holds:

- $d(T_i, T_j) \geq 1 - \lambda$ for $1 \leq i < j \leq k$,
- $d(A_i, A_j) \geq 1 - \lambda$ for $1 \leq i < j \leq k$,
- $d(A_i, T_j) \geq 1 - \lambda$ for $1 \leq i \leq j \leq k$,
- $d(T_i, A_j) \geq 1 - \lambda$ for $1 \leq i < j \leq k$,

- $|A_i| \geq c|T|$ for $i = 1, 2, \dots, k, k+1$, and
- $|T_i| \geq c|T|^\epsilon$ for $i = 1, 2, \dots, k$.

We refer to the sets T_1, \dots, T_k from the m -sequence as *transitive sets of the m -sequence*. We say that a m -sequence of length $2k+1$ is M -big if $T_i \geq M$ for $i = 1, 2, \dots, k$.

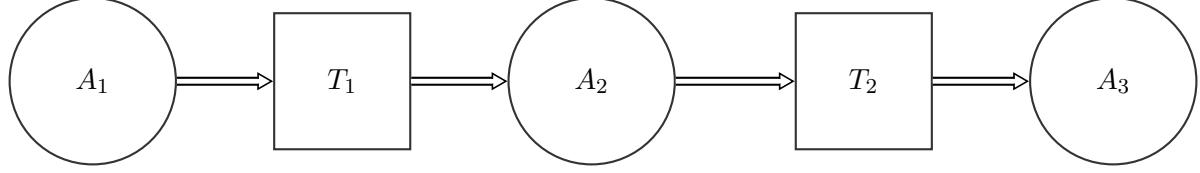


Fig.4 Schematic representation of the (c, λ, ϵ) - m -sequence. This sequence consists of three linear sets: A_1, A_2, A_3 and two transitive sets: T_1 and T_2 . The arrows indicate the orientation of most of the edges going between different elements of the (c, λ, ϵ) - m -sequence. Each T_i satisfies: $|T_i| \geq cn^\epsilon$ and each A_i satisfies: $|A_i| \geq c \cdot n$, where $n = |T|$.

For the pairwise disjoint subsets $A_1, \dots, A_{k+1}, T_1, \dots, T_k \subseteq V(T)$, where T_1, \dots, T_k are transitive, we say that a sequence $(A_1, T_1, \dots, A_k, T_k, A_{k+1})$ is a *smooth (c, λ, ϵ) - m -sequence of length $2k+1$* if the following holds:

- sequence (T_1, \dots, T_k) is a smooth (c, λ, ϵ) - t -sequence
- sequence (A_1, \dots, A_{k+1}) is a smooth (c, λ) - l -sequence
- $d(\{v\}, T_j) \geq 1 - \lambda$ for $1 \leq i \leq j \leq k$ and $v \in A_i$,
- $d(A_i, \{v\}) \geq 1 - \lambda$ for $1 \leq i \leq j \leq k$ and $v \in T_j$,
- $d(\{v\}, A_j) \geq 1 - \lambda$ for $1 \leq i < j \leq k$ and $v \in T_i$,
- $d(T_i, \{v\}) \geq 1 - \lambda$ for $1 \leq i < j \leq k$ and $v \in A_j$,
- $|A_i| \geq c|T|$ for $i = 1, 2, \dots, k, k+1$, and
- $|T_i| \geq c|T|^\epsilon$ for $i = 1, 2, \dots, k$.

Note that every smooth (c, λ, ϵ) - m -sequence is a (c, λ, ϵ) - m -sequence.

If a smooth (c, λ, ϵ) - m -sequence $(A_1, T_1, \dots, A_k, T_k, A_{k+1})$ satisfies:

- $d(T_i, T_j) = 1$ for $i, j \in \mathcal{I}$, where $i < j$ and $\mathcal{I} \subseteq \{1, 2, \dots, k\}$,

then we say that $(A_1, T_1, \dots, A_k, T_k, A_{k+1})$ is an \mathcal{I} -strong (c, λ, ϵ) - m -sequence of length $2k+1$.

Whenever we do not care about parameters of the (c, λ, ϵ) - t -sequences, (c, λ) - l -sequences or (c, λ, ϵ) - m -sequences under consideration, we refer to them simply as: t -sequences, l -sequences and m -sequences respectively.

For two disjoint subsets $A_1, T_1 \subseteq V(T)$ such that T_1 is transitive we say that a pair (A_1, T_1) is (c, ϵ) -saturated if the following holds:

- $|A_1| \geq c|T|$,
- $|T_1| \geq c|T|^\epsilon$, and
- $d(A_1, T_1) = 1$ or $d(T_1, A_1) = 1$.

4 Quasi-polynomial algorithm for coloring H -free tournaments for a constellation H

4.1 Introduction

Whenever we consider algorithms involving tournaments, we assume that the input is a tournament description given by adjacency lists. This section presents two main algorithms (and several subroutines used by them): one that finds a polynomial-size transitive subtournament in the strong m -sequence of an H -free tournament (algorithm *PolyTrans*) and one that colors an H -free tournament (algorithm *Color- H -free*), where H is a constellation. The former is in fact used to construct the coloring produced by the latter one. The latter one takes as an input only forbidden constellation H and H -free tournament T . The core part of the algorithm *PolyTrans* is the recursive algorithm *PolyTransCore* that gets as an input colored m -sequence (the coloring is done in the initial phase of the algorithm *PolyTrans*) and outputs a polynomial-size transitive subset.

We would like to give a general idea of our algorithmic approach first since the algorithm is complicated. Assuming that T is an H -free tournament, we find in T a long enough sequence of big linear and transitive sets such that most of the edges between those sets go from these sets that are earlier in the sequence to those that are placed later. At that point no assumption about the structure of H is required. We handle that part of the algorithm without the use of the regularity lemma and that enables us to significantly improve best known lower bounds on $\epsilon(H)$. In the second stage we heavily use the fact that H , as a constellation, has a specific ordering of vertices. The algorithm finding polynomial-size transitive subtournament uses the following technical procedures: *Find-L-Sequence*, *Find-Clique*, *Find-M-Sequence*, *MakeSmooth*, *FindStrong-M-Sequence*. Algorithm *Find-L-Sequence* finds a (c, λ) - l -sequence in an H -free tournament. Algorithm *Find-Clique* finds a clique in a dense k -partite undirected graph. Algorithm *Find-M-Sequence* is responsible for finding m -sequences in H -free tournaments. Algorithm *MakeSmooth* makes them smooth. Finally, algorithm *FindStrong-M-Sequence* constructs a strong m -sequence in an H -free tournament. That sequence is the input for the *PolyTrans* algorithm. This short summary will be clearer later when we describe all the algorithms in detail.

Let H be a constellation with $h = |H|$. Let θ be a constellation ordering of H . We define a function $\zeta : V(H) \rightarrow \mathbb{N}$ as follows:

- if $v \in V(H)$ is a leaf of a star let $\zeta(v) = i(2h + 1) + 1$, where v is the i th leaf under ordering θ ,
- otherwise let $\zeta(v) = j(2h + 1) + 2r$, where v is the r th center after the j th leaf and before the $(j + 1)$ th leaf under the ordering θ .

First we show algorithms: *PolyTrans* and *Color- H -free*. Then we will describe all supportive procedures mentioned above.

4.2 An overview of the method

The constructive proof we are about to present that all the constellations satisfy the conjecture is very technical. Therefore before going into details we would like to explain main steps of the proof. In this subsection we give reader the intuition how the proof works, what are the most important parts of the proof, finally - why the ideas used to prove the conjecture for galaxies are not sufficient to succeed with constellations and what are the new techniques that need to be used in this setting.

Notice first that we can always assume that we have a sequence of linear sets and big transitive subtournaments such that for any two of them the one that appears first in the sequence is almost

complete to the one that appears later. This sequence is what we call a (c, λ, ϵ) - m -sequence. Parameter c encodes lower bounds on the sizes of linear and transitive elements. Parameter ϵ specifies the value of the exponent in the lower bound on the size of the transitive element. Finally, parameter λ specifies lower bound on the directed density between different elements of the sequence. By "almost adjacent" we mean that the directed density is very close to 1. Existence of such a sequence is an immediate consequence of the regularity lemma, but since we do not want to use that tool, we prove that fact in a very different way. This enables us to get a bound on the EH coefficient that can be expressed in the compact way. What is important here is that at this point we did not need to assume any specific structure of the tournament H that is being excluded. If we need to summarize the proof of the conjecture for galaxies from [5] in one sentence, we should say: "linear sets for centers of stars, transitive subtournaments for leaves". We give a proof by contradiction, assume that an H -free tournament does not have polynomial-size transitive subtournaments and construct a copy of H in it. The copy will be constructed star by star. We proceed by creating first an appropriate (c, λ, ϵ) - m -sequence mentioned above. Then we are looking for stars with centers in linear sets and leaves in transitive sets. When the star is found it is removed and the entire sequence is updated. The update is done in such a way that we can simply merge a star we have just found with the remaining part of the tournament H (that will be found in the new sequence) to get a copy of H .

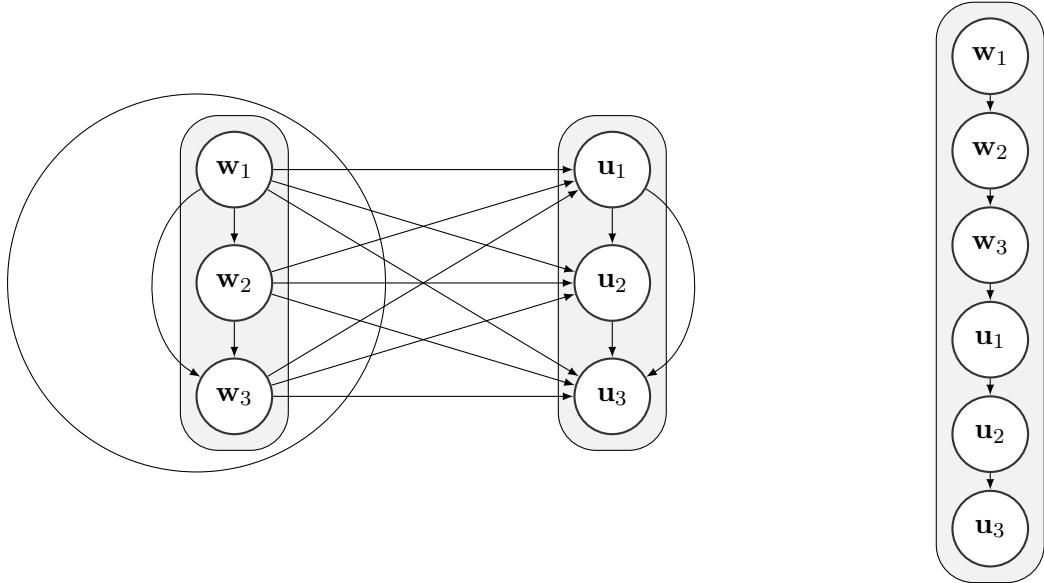


Fig.5 The method to construct polynomial-size transitive subtournament. A circle represents a linear set and $\{w_1, \dots, w_3\}$ is a big transitive subset that (by induction) can be extracted from it. A set $\{u_1, \dots, u_3\}$ represents another transitive set of substantial size. Set $\{w_1, \dots, w_3\}$ is complete to $\{u_1, \dots, u_3\}$. Thus $\{w_1, \dots, w_3, u_1, \dots, u_3\}$ is also a transitive set. For $\epsilon > 0$ small enough this set is of size at least n^ϵ , where n is the size of the H -free tournament T .

If the star cannot be found, then a simple argument using Pigeonhole principle shows that we must have a substantial transitive subset complete to/from a linear set (see: Fig. 5). The key observation here is that a transitive set of the substantial size complete to or from a linear set gives us a polynomial-size transitive subtournament. The crucial element that makes this method work is that whenever we are looking for a star there is no need to take care of the right type of adjacency between leaves if all candidates for them were chosen from the same transitive set.

The right type of adjacency is given for granted. The price that is paid is the fact that we cannot allow centers of stars to be between leaves of another star since that would require looking for centers also in transitive chunks. That in turn will not enable us to get a polynomial-size transitive subtournament. This was also the main reason why the authors started to work on more general techniques that would handle cases where centers of stars are between leaves of another stars, i.e. more general configurations of stars. The conceptual idea of this more general technique is to change the paradigm: "centers in linear sets, leaves in transitive subtournaments" by starting with the (c, λ, w) - t -sequence that consists only of big transitive chunks (by big we mean of size at least $c \cdot |T|^\epsilon$ for some constant $c, \epsilon > 0$).

We start proceeding as in the galaxy-proof, but try to use t -sequences first (in particular we look for centers of stars in transitive chunks, see Fig. 6).

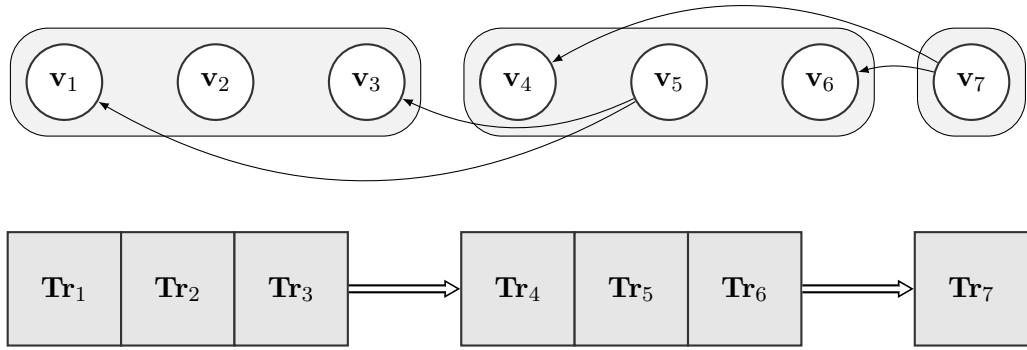


Fig.6 Constellation H with its constellation ordering of vertices (above) and the associated t -sequence χ . Sequence χ consists of three transitive sets, first two are partitioned into three equal-length subsets. There is a 1-1 mapping between vertices of H and subchunks Tr_i . The goal is to look for a node v_i in Tr_i .

The problem we face (that was mentioned by us before) is that now we do not necessarily get as an outcome a big linear set complete to or from a big transitive chunk. Instead, we obtain two transitive chunks such that one of them is complete to the other one. However those chunks, even after merging, may not give big enough transitive subtournament. This is the place where we need to use strong (c, λ, ϵ) - m -sequences.

We repeat our previous procedure several times in many different regions of the long enough t -sequence. By doing it, using Ramsey argument, we can conclude that we get arbitrarily large set of transitive subchunks of the elements of the sequence with the additional property that the subchunks appearing earlier in the sequence are complete to those appearing later (altogether they may still not give a big enough transitive subtournament). We may also assume without loss of generality that between those subchunks we have big linear sets such that each linear set is almost complete from all subchunks preceding it and almost complete to all subchunks that it precedes. This can be easily done if we slightly enrich the t -sequence we started with by introducing linear sets between transitive chunks. This can be always achieved since we have already observed that we can start with the arbitrarily long (c, λ, ϵ) - m -sequence. But now we can again try to build a constellation star by star. Notice however that in contrast to the previous scenario, we can look for different leaves of the same star in different transitive chunks (see Fig. 7). This is possible because for any two transitive chunks of the strong sequence the one appearing earlier in the sequence is complete to the one appearing later. Therefore while constructing a star in such a way that leaves are being found in transitive chunks we have the right type of adjacency between those candidates for leaves for granted. This is no longer true if the structure we are given is no strong.

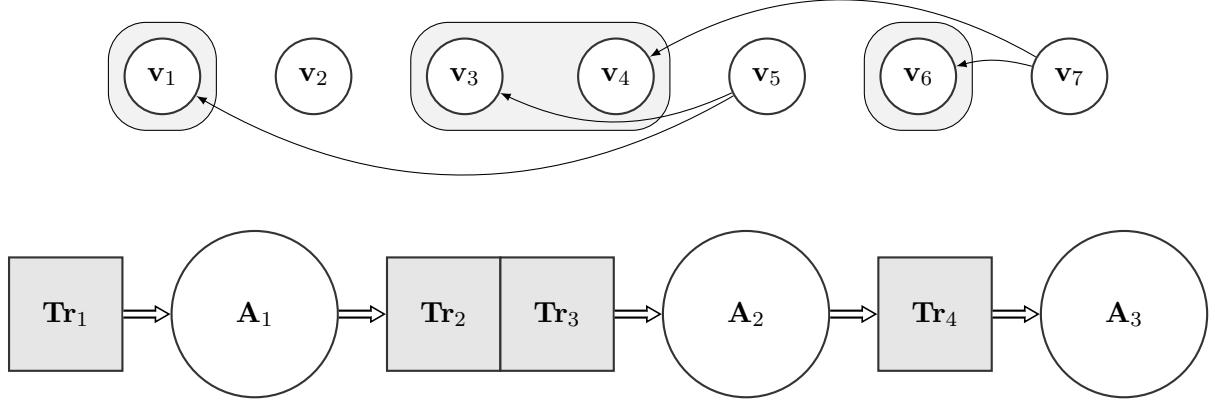


Fig.7 Constellation H with its constellation ordering of vertices (above) and the associated section of the (c, λ, ϵ) - m -sequence χ corresponding to H . This section of the sequence χ consists of three transitive sets and three linear sets. The second transitive set is partitioned into two equal-length subsets. Centers of stars as well as singletons (v_2, v_5, v_7) are being looked for in the linear sets (A_1, A_2, A_3). Leaves of stars are being looked for in the transitive sets Tr_i .

We cannot get a strong (c, λ, w) - m -sequence immediately. It can be constructed if the first approach that we used to find a copy of H or a polynomial-size transitive subtournament fails. If at any stage of our analysis in the second part of the algorithm (when we operate on the strong m -sequence) some star cannot be constructed then we get a linear set complete to/from a big transitive chunk. But now that gives us polynomial-size transitive subtournament and the proof is completed. Note that even though we significantly relax the condition we put on the configurations of stars by introducing the family of constellations, we still cannot look for a center of the star and some of its leaves in the same transitive chunk. This is the case since obviously no backward edges can be found in a transitive chunk. Therefore there are still some limitations put on the configuration of stars. However, as mentioned before, there are much weaker than previously. We are ready now to go into technical details of our algorithm.

4.3 Algorithm *PolyTrans*

We start with the *PolyTrans* algorithm that takes as an input an \mathcal{I} -strong N -big (c, λ, ϵ) - m -sequence of the H -free tournament T and outputs a transitive subtournament of T of size at least $|T|^\epsilon$, where $\epsilon = \frac{\log(1-\hat{c})}{\log(\hat{c})}$ for $\hat{c} = \frac{c}{2^{7h^2}}$, as long as $\lambda \leq \frac{1}{2^{25h^2}h}$, $n = |T| \geq \frac{2^{21h^2}}{c}$ and $N \geq 2^{21h^2}$. What is also given as a part of the input for *PolyTrans* is the procedure \mathcal{P} which, for every subtournament T_S of T other than T , computes a transitive subtournament of T_S of order at least $|T_S|^\epsilon$. Assume that $V(H) = \{1, 2, \dots, h\}$ and that $(1, 2, \dots, h)$ is a constellation ordering of H .

Algorithm 4.1 (*Algorithm PolyTrans returning a transitive subtournament of a polynomial-size*)

- **Input:** An \mathcal{I} -strong (c, λ, ϵ) - m -sequence of length $k = 2h^2 + 4h + 1$ for $\mathcal{I} = \{h + 1, 2h + 1, \dots, h^2 + 1\}$ in an H -free tournament T of order $n \geq \frac{2^{21h^2}}{c}$. The sequence is N -big for $N \geq 2^{21h^2}$. Tournament H is a constellation. It is assumed that: $\lambda \leq \frac{1}{2^{25h^2}h}$ and $\epsilon = \frac{\log(1-\hat{c})}{\log(\hat{c})}$ for $\hat{c} = \frac{c}{2^{7h^2}}$. Procedure \mathcal{P} is given that for every subtournament T_S of T different than T computes a transitive subtournament of T_S of order at least $|T_S|^\epsilon$. It is assumed that this procedure runs in time $h(|T_S|)$ for some given function h .

- **Output:** A transitive subtournament of T of size at least $|T|^\epsilon$.
- **Description:** Initialize σ to be the set of all the stars of H . We first run algorithm *MakeSmooth* (with $f = 0$) on the m -sequence from the input to make the sequence smooth. Algorithm *MakeSmooth* outputs a smooth $(\frac{c}{2}, 4\lambda, \epsilon)$ - m -sequence which we denote as: (C_1, \dots, C_k) (see: description of *MakeSmooth*). We give colors to the vertices of (C_1, \dots, C_k) as follows:

- if $C_i = C_{\zeta(v)}$ for some $v \in V(H)$ and i is even, i.e. C_i is transitive, then color the first $\frac{|C_i|}{3}$ vertices of C_i in the transitive ordering by v and the rest by $h + 1$,
- if $C_i = C_{\zeta(v)}$ for some $v \in V(H)$ and i is odd then color an arbitrary subset of $\frac{|C_i|}{3}$ vertices of C_i by v and the rest by $h + 1$,
- otherwise color all vertices of C_i by $h + 1$.

Then run algorithm *PolyTransCore* (see: below) on the colored smooth $(\frac{c}{2}, 4\lambda, \epsilon)$ - m -sequence described above for parameter $\xi = \frac{1}{6}$, procedure \mathcal{P} and a set of stars σ .

Algorithm 4.2 (*Algorithm PolyTransCore*)

- **Input:** A (c, λ, ϵ) - m -sequence (C_1, \dots, C_k) of the H -free tournament T , where vertices of C_i s are colored by the set $\{1, 2, \dots, h + 1\}$, a set of stars σ and parameter $\xi > 0$. It is assumed that if there exists a vertex of C_i colored by some color j then at least $\xi|C_i|$ vertices of C_i are colored by that color. Procedure \mathcal{P} is given that for every subtournament T_S of T different than T computes a transitive subtournament of T_S of order at least $|T_S|^\epsilon$.
- **Output:** A transitive subtournament of T of size at least $|T|^\epsilon$.
- **Description:** Take a star $\Sigma^* \in \sigma$. Without loss of generality assume that it is a left star. Denote by c its center and by y_1, \dots, y_q its leaves. Let X be the set of vertices of $C_{\zeta(c)}$ that are colored by c and let \mathcal{T}^i be the set of vertices of $C_{\zeta(y_i)}$ that are colored by y_i for $i = 1, 2, \dots, q$. Assume first that there does not exist a vertex $\tau \in X$ and vertices r_1, \dots, r_q such that $r_i \in \mathcal{T}^i$ and τ is adjacent from all r_1, \dots, r_q . Then, by the Pigeonhole Principle, at least $\frac{1}{q}|X|$ vertices of X , call this set \mathcal{C} , are adjacent to all vertices of \mathcal{T}^{j^*} for some $j^* \in \{1, 2, \dots, q\}$. Then run procedure \mathcal{P} on $T|\mathcal{C}$ and merge the transitive subtournament output by the procedure with \mathcal{T}^{j^*} to get a bigger transitive subtournament T^b . Output T^b . Assume now that there exists a vertex $\tau \in X$ and vertices r_1, \dots, r_q such that $r_i \in \mathcal{T}^i$ for $i = 1, 2, \dots, q$ and τ is adjacent from all r_1, \dots, r_q . Let $D_{i,j}$ be a subset of vertices from C_i , other than $X, \mathcal{T}^1, \dots, \mathcal{T}^q$, colored by some fixed color j . Denote by $N_0^{D_{i,j}} \subseteq D_{i,j}$ the subset of $D_{i,j}$ consisting of vertices of $D_{i,j}$ adjacent from τ if τ is before $D_{i,j}$ in the m -sequence and adjacent to τ otherwise (the ordering in the m -sequence is induced by an ordering of the sets C_i and a transitive ordering within transitive parts of the m -sequence). Similarly, denote by $N_k^{D_{i,j}} \subseteq D_{i,j}$ for $k = 1, 2, \dots, q$ the subset of $D_{i,j}$ consisting of vertices of $D_{i,j}$ adjacent from r_i if r_i is before $D_{i,j}$ in the m -sequence and adjacent to r_i otherwise. Denote $N^{D_{i,j}} = N^{D_{i,j}} \cap N_0^{D_{i,j}}$, where $\hat{N}^{D_{i,j}} = \bigcap_{k=1,2,\dots,q} N_k^{D_{i,j}}$. We run algorithm *MakeSmooth* with parameter f satisfying: $(1 - f) = \xi(1 - \frac{\lambda h}{\xi})$ on the given m -sequence (C_1, \dots, C_k) , where we take: $S_i^j = N^{D_{i,j}}$. We obtain new smooth m -sequence that we denote as χ . In this new m -sequence the coloring is inherited from the old one. We delete Σ^* from σ . Now we rerun algorithm *PolyTransCore* on χ with the updated set of stars and updated parameters c, λ, ξ : $c \rightarrow \frac{c\xi}{2}(1 - \frac{\lambda h}{\xi})$, $\lambda \rightarrow \frac{4\lambda k}{\xi^2(1 - \frac{\lambda h}{\xi})^2}$, $\xi \rightarrow \frac{\xi}{2}(1 - \frac{\lambda h}{\xi})$.

The naive implementation of *PolyTrans* algorithm runs in time $h((1 - c)n) + \text{poly}(n)$, where $\text{poly}(n)$ is an expression polynomial in n . This is true since whenever procedure \mathcal{P} is called it is run on a tournament of at most $(1 - c)n$ vertices. The correctness of the algorithm *PolyTrans* is proven later.

4.4 Algorithm *Color-H-free*

We are ready to give our main algorithm that for a given constellation H with $|H| = h$, properly colors every H -free tournament T using at most $|T|^{1 - \frac{1}{2^{250h^2+1}}} \log(|T|)$ colors and runs in time $e^{O(\log(n)^2)}$. As a byproduct, the algorithm finds in T a transitive subtournament of order at least at least $|T|^{\frac{1}{2^{250h^2+1}}}$. This tournament is the first color class constructed by the algorithm on the way to produce the entire coloring.

Algorithm 4.3 (*Algorithm Color-H-free coloring H -free tournaments, where H is a constellation*)

- **Input:** Constellation H and H -free tournament T .
- **Output:** Proper coloring of T that uses at most $|T|^{1 - \frac{1}{2^{250h^2+1}}} \log(|T|)$ colors.
- **Description:** Run algorithm *Find-L-Sequence* with $\lambda = \frac{1}{2^{29h}}$ and $k = 2h + 3$ to get a l -sequence \mathcal{L} . Truncate the constructed l -sequence \mathcal{L} by deleting its last element. Note that what we get is another l -sequence \mathcal{L}^t . Give it as an input to algorithm *Find-M-Sequence*. As a procedure *Sub* needed by *Find-M-Sequence* (see the description of the algorithm in the next subsection) use algorithm *Color-H-free* itself (thus we use a recursive call of the algorithm *Color-H-free* on the smaller graph). Denote the output m -sequence by \mathcal{M} . Truncate it (by deleting its last elements) to reduce its length to exactly $2^{h+2}(h+1) + 2h + 1$. Denote the truncated m -sequence as \mathcal{M}^t . Now run on \mathcal{M}^t algorithm *FindStrong-M-Sequence* to get an \mathcal{I} -strong m -sequence \mathcal{M}^s . Give \mathcal{M}^s as an input to algorithm *PolyTrans*. Again, as a procedure \mathcal{P} needed by *FindStrong-M-Sequence* use algorithm *Color-H-free* itself. As an output we obtain a transitive subtournament T_1 . This transitive tournament becomes a first element of the output of the algorithm *Color-H-free* (remember that *Color-H-free* outputs both: polynomial-size transitive subset and the coloring). The remaining element of the output is the coloring and we are about to give it. Color all vertices of T_1 by 1. Remove T_1 from T and repeat the entire procedure to obtain a transitive subtournament T_2 . Color all its vertices by 2. Remove T_2 from T and keep repeating the procedure. Continue until the tournament you are left with is empty. By that time all the vertices of T were colored and one can easily note that this is a proper coloring of T . The color classes are the sets: $V(T_1), V(T_2), \dots$. Output that coloring.

4.5 Technical algorithms

4.5.1 Introduction

In this section we present algorithms: *Find-L-Sequence*, *Find-Clique*, *Find-M-Sequence*, *MakeSmooth*, *FindStrong-M-Sequence* that serve as technical subroutines for algorithms introduced earlier.

4.5.2 Algorithm *Find-L-Sequence*

For a given $0 < \lambda < 1$ and $k \geq 0$ let $\lambda_i = (\frac{\lambda^2}{4^k h^{4k}})^{2^i h^{2i}}$ for $i = 0, 1, \dots, k$.

We start by describing *Find-L-Sequence* algorithm that calculates a (c, λ) -l-sequence of length 2^k for some given $k \geq 0$ in the H -free tournament T with: $|T| \geq \frac{2^{k+1}(h+1)h^{2k}}{\lambda_k^{hk}}$, where H is an arbitrary tournament with $|H| \geq 2$, $V(H) = \{v_1, \dots, v_h\}$, and $c = c(H, k, \lambda) = \frac{\lambda_k^{|H|k}}{2^k |H|^{2k}}$. The algorithm *Find-L-Sequence* uses subroutine *Find-L-Sequence-Core* which recursively runs algorithm *Find-L-Sequence* on the smaller input.

Algorithm 4.4 (*Algorithm Find-L-Sequence constructing a (c, λ) -l-sequence in an H -free tournament*)

- **Input:** (λ, k, H, T) , where T is H -free, $|H| \geq 2$.
- **Output:** A (c, λ) -l-sequence of length 2^k in H , where $c = c(H, k, \lambda) = \frac{\lambda_k^{hk}}{2^k h^{2k}}$.
- **Description:** If $k = 0$ then output $V(T)$. Assume now that $k > 0$. Then choose in T arbitrarily h pairwise disjoint sets: S_1, \dots, S_h , each of size $\lfloor \frac{n}{h+1} \rfloor$, where $n = |T|$. Run subroutine *Find-L-Sequence-Core* with parameters: $(H, \lambda, \lambda_k, k, H, T|S_1 \cup \dots \cup S_h, S_1, \dots, S_h)$, where $h = |H|$.

Subroutine 4.1 (*Subroutine Find-L-Sequence-Core*)

- **Input:** $(H, \lambda, \lambda_k, k, H^r, T^r, S_{t_1}, \dots, S_{t_{|H^r|}})$, where $k > 0$, $|H| > 1$, $V(H^r) = \{v_{t_1}, \dots, v_{t_{|H^r|}}\}$ for some vertices: $v_{t_1}, \dots, v_{t_{|H^r|}}$ and T^r is H -free. It is assumed that: $S_{t_1}, \dots, S_{t_{|H^r|}}$ are nonempty.
- **Output:** Sequence of pairwise disjoint subsets of $V(T)$ of length 2^k (see: Description below).
- **Description:** If $|H^r| = 1$ then raise an exception (in the analysis of the algorithm we will show that the exception in fact will never be raised). Now assume that $|H^r| > 1$. Denote $h^r = |H^r|$. For a vertex $v \in S_{t_1}$ and an index $j = 2, 3, \dots, h^r$ denote by $N_j(v)$ the set of vertices of S_{t_j} adjacent from v if v_{t_j} is adjacent from v_{t_1} or adjacent to v if v_{t_j} is adjacent to v_{t_1} in H^r . Calculate values $|N_j(v)|$ for every $v \in S_{t_1}$ and $j = 2, 3, \dots, h^r$.

If there exists a vertex $v^* \in S_{t_1}$ such that $|N_j(v^*)| \geq \lambda_k |S_{t_j}|$ for every $j \in \{2, 3, \dots, h^r\}$ then run recursively subroutine *Find-L-Sequence-Core* with parameters:

$(H, \lambda, \lambda_k, k, H^r \setminus \{v_{t_1}\}, T^r | N_2(v^*) \cup \dots \cup N_{h^r}(v^*), N_2(v^*), \dots, N_{h^r}(v^*)).$

If this is not the case then (by Pigeonhole Principle) at least $\frac{|S_{t_1}|}{h^r - 1}$ vertices v of S_{t_1} satisfy the following: there exists $j^* \in \{2, 3, \dots, h^r\}$ such that $|N_{j^*}(v)| < \lambda_k |S_{t_{j^*}}|$. Denote this set of vertices by W . Run algorithm *Find-L-Sequence* with parameters: $(\lambda, k - 1, H, T^r | W)$ to get a l-sequence: $(A_1, \dots, A_{2^{k-1}})$. Run algorithm *Find-L-Sequence* with parameters: $(\lambda, k - 1, H, T^r | S_{t_{j^*}})$ to get a l-sequence: $(A'_1, \dots, A'_{2^{k-1}})$. Note that $d(W, S_{t_{j^*}}) \geq 1 - \lambda_k$ or $d(S_{t_{j^*}}, W) \geq 1 - \lambda_k$. If the former is true then output the sequence $(A_1, \dots, A_{2^{k-1}}, A'_1, \dots, A'_{2^{k-1}})$. Otherwise output the sequence $(A'_1, \dots, A'_{2^{k-1}}, A_1, \dots, A_{2^{k-1}})$.

Algorithm *Find-L-Sequence* runs in polynomial time. Its correctness will be proven later. Later we will also analyze its running time in more detail.

4.5.3 Algorithm *Find-Clique*

For an undirected graph G and two nonempty disjoint sets: $A, B \subseteq V(G)$ we define, by an analogy to the directed setting: $d(A, B) = \frac{e_{A,B}}{|A||B|}$, where $e_{A,B}$ is the number of edges between A and B . Assume now that we have a k -partite undirected graph with color classes: V_1, \dots, V_k . Assume besides that for every $1 \leq i < j \leq k$ the following holds: $d(V_i, V_j) \geq 1 - \lambda$, where: $\lambda \leq \frac{1}{3^{2k+1}k}$. Now we will present an algorithm *Find-Clique* that finds in this graph a clique: $\{v_1, \dots, v_k\}$ such that $v_i \in V_i$ for $i = 1, 2, \dots, k$.

Algorithm 4.5 (*Algorithm Find-Clique finding a clique in a dense k -partite undirected graph*)

- **Input:** k -partite undirected graph with color classes: V_1, \dots, V_k such that for every $1 \leq i < j \leq k$ the following holds: $d(V_i, V_j) \geq 1 - \lambda$, where λ satisfies: $0 < \lambda < \frac{1}{3^{2k+1}k}$.
- **Output:** A clique $\{v_1, \dots, v_k\}$ such that $v_i \in V_i$ for $i = 1, 2, \dots, k$.
- **Description:** If $k = 1$ then output an arbitrary vertex of V_1 . Now assume that $k > 1$. Define $W_i = \{v \in V_1 : d(\{v\}, V_i) < 1 - 2k\lambda\}$ for $i = 2, 3, \dots, k$. For every i each set W_i may be easily computed. Having sets W_i , take a vertex $v_1 \in V_1 \setminus (W_2 \cup \dots \cup W_k)$. For every $i = 2, 3, \dots, k$ compute $V'_i = V_i \cap N_i^v$, where N_i^v is the set of vertices adjacent to v_1 in V_i . Run recursively algorithm *Find-Clique* on the $(k-1)$ -partite induced subgraph with color classes: V'_2, \dots, V'_k , and update λ : $\lambda \rightarrow \frac{\lambda}{(1-2k\lambda)^2}$. You obtain a set $\{v_2, \dots, v_k\}$. Output the set $\{v_1, \dots, v_k\}$.

The algorithm clearly runs in $\text{poly}(\max_{i=1, \dots, k} |V_i|)$ time (note that k is a constant). Its correctness will be proven later.

4.5.4 Algorithm *Find-M-Sequence*

Let T be a tournament with n vertices. Assume now that (A_1, \dots, A_{2k+1}) is a (c, λ) - l -sequence in T , where $\lambda \leq \frac{\Lambda}{4(2k+1)3^{4k+3}}$. Assume furthermore that there exists a subroutine *Sub* that for every $i = 1, 2, \dots, k$ and for every subtournament T_S of A_{2i} computes in time $O(h(|T_S|))$ a transitive subtournament of T_S of size at least $|T_S|^\epsilon$ for some given $\epsilon > 0$. Under these conditions we will show an algorithm *Find-M-Sequence* that computes in T a (c', Λ, ϵ) - m -sequence of length $2k+1$ for $c' = \min((\frac{c}{2})^\epsilon, c)$ which in addition is $(\log(cn) - 2)$ -big. From the characteristic of the sequence we already know that the sequence is $c'n$ -big and asymptotically for large n this is a stronger property than being $(\log(cn) - 2)$ -big. However for the corner cases for small n (that we will need for induction to prove that the algorithm produces a transitive subset in the H -free tournament of the desired size) we will also need $(\log(cn) - 2)$ -bigness property.

Algorithm 4.6 (*Algorithm Find-M-Sequence finding m -sequences*)

- **Input:** A (c, λ) - l -sequence (A_1, \dots, A_{2k+1}) in T for $\lambda \leq \frac{\Lambda}{4(2k+1)3^{4k+3}}$ for some parameter $0 < \Lambda < 1$ and a subroutine *Sub* that for every $i = 1, 2, \dots, k$ and for every subtournament T_S of A_{2i} computes in time $O(h(|T_S|))$ a transitive subtournament of T_S of size at least $|T_S|^\epsilon$ for some given $\epsilon > 0$.
- **Output:** A $(\min((\frac{c}{2})^\epsilon, c), \Lambda, \epsilon)$ - m -sequence of length $2k+1$ in T which is $(\log(cn) - 2)$ -big.

- **Description:** Denote $n = |T|$. Let $i \in 1, 2, \dots, k$. Compute a transitive subset $Tr_1^i \subseteq A_{2i}$ of size at least $\log(\frac{cn}{2}) - 1$ for $i = 1, 2, \dots, k$. Such a subset always exists since $|A_{2i}| \geq \frac{cn}{2}$ (it also can be efficiently computed, we will discuss this in more detail later). Now use subroutine Sub to compute a transitive subset $Tr_2^i \subseteq A_{2i}$ of size $\lceil (\frac{c}{2})^\epsilon n^\epsilon \rceil$. Remove from A_{2i} the bigger subset from the set: $\{Tr_1^i, Tr_2^i\}$ and denote this bigger one by T_1^i . Continue until the set you are left with is of size smaller than $\frac{|A_{2i}|}{2}$. Denote transitive subsets obtained in such a way as: $T_1^i, \dots, T_{r_i}^i$. Construct a $(2k+1)$ -partite graph G with color classes: V_1, \dots, V_{2k+1} as follows:

- $V_{2i+1} = \{A_{2i+1}\}$ for $i = 1, 2, \dots, k$,
- $V_{2i} = \{T_1^i, \dots, T_{r_i}^i\}$ for $i = 1, 2, \dots, k$,
- make two vertices $X \in V_a, Y \in V_b$ for $1 \leq a < b \leq 2k+1$ adjacent if $d(X, Y) \geq 1 - \Lambda$.

Run algorithm *Find-Clique* on G to obtain a sequence (Y_1, \dots, Y_{2k+1}) that induces a clique in G ($Y_i \in V_i$ for $i = 1, 2, \dots, 2k+1$). Output (Y_1, \dots, Y_{2k+1}) .

Denote $A^m = \max_{i=1, \dots, 2k+1} |A_i|$. The algorithm runs in time $O(h(A^m))n^{1-\epsilon} + \text{poly}(n)$, where $\text{poly}(n)$ is a polynomial factor. Its correctness and running time will be proven later.

4.5.5 Algorithm *MakeSmooth*

Assume that we are given a sequence (C_1, \dots, C_k) that is either a (c, λ, ϵ) - m -sequence, a (c, λ, ϵ) - t -sequence or an (c, λ) - l -sequence. Assume that $S_i^j \subseteq C_i$ for $j = 1, 2, \dots, s_i$. Assume furthermore that S_i^j 's are pairwise disjoint for $j = 1, 2, \dots, s_i$ and $|S_i^j| \geq (1-f)|C_i|$ for $i = 1, 2, \dots, k, j = 1, 2, \dots, s_i$, where $0 < f < 1$ is some fixed parameter. Denote $L = s_1 + \dots + s_k$. We will show an algorithm that extracts a subset from every S_i^j and uses extracted subsets to construct a smooth (c', λ', ϵ) - m -sequence, a smooth (c', λ', ϵ) - t -sequence or a smooth (c', λ') - l -sequence respectively, where $c' = \frac{c}{2}(1-f)$ and $\lambda' = \frac{4\lambda L}{(1-f)^2}$. Each element of the constructed sequence (m -sequence, t -sequence or l -sequence) is the union of the corresponding extracted subsets. The goal is to make an input sequence smooth by taking subsets, but in such a way that a significant fraction of elements from each part of the partition defined by sequences S_i^j is used in the built smooth sequence.

The naive implementation of the algorithm below clearly runs in polynomial time. Its correctness is proven in the next section.

Algorithm 4.7 (Algorithm *MakeSmooth*)

- **Input:** A (c, λ, ϵ) - m -sequence, a (c, λ, ϵ) - t -sequence or a (c, λ) - l -sequence (C_1, \dots, C_k) , a set of subsets S_i^j for $i = 1, 2, \dots, k, j = 1, 2, \dots, s_i$ such that $S_i^j \subseteq C_i, |S_i^j| \geq (1-f)|C_i|$ and S_i^j are pairwise disjoint for $i = 1, 2, \dots, k, j = 1, 2, \dots, s_i$ and some $0 < f < 1$. It is assumed that $s_i = 1$ for $i = 1, 2, \dots, k$ if (C_1, \dots, C_k) is a l -sequence, and $s_i = 1$ for $i = 1, 3, 5, \dots$ if (C_1, \dots, C_k) is a m -sequence (in other words, if $s_i > 1$ then i corresponds to the transitive set in (C_1, \dots, C_k)). Denote $L = s_1 + \dots + s_k$.
- **Output:** A smooth (c', λ', ϵ) - m -sequence, a smooth (c', λ', ϵ) - t -sequence or a smooth (c', λ') - l -sequence (C'_1, \dots, C'_k) respectively, where $c' = \frac{c}{2}(1-f)$, $\lambda' = \frac{4\lambda L}{(1-f)^2}$ and $C'_i \subseteq \bigcup_{j=1,2,\dots,s_i} S_i^j$ for $i = 1, 2, \dots, k$. Besides we have: $|C'_i \cap S_i^j| \geq \frac{cn}{2}(1-f)$ for $i = 1, 2, \dots, k, j = 1, 2, \dots, s_i$.

- **Description:** Let us assume that (C_1, \dots, C_k) is a (c, λ, ϵ) - m -sequence. For two remaining cases the algorithm is completely analogous. For $i, j \in \{1, 2, \dots, k\}$, $i \neq j$, $t_1 \in \{1, 2, \dots, s_i\}$, $t_2 \in \{1, 2, \dots, s_j\}$ denote by $C_{i,t_1}^{j,t_2} \subseteq S_i^{t_1}$ the set of vertices of $S_i^{t_1}$ that:

- are adjacent to at least $(1 - \frac{2L\lambda}{(1-f)^2})|S_j^{t_2}|$ vertices of $S_j^{t_2}$ if $i < j$ or
- are adjacent from at least $(1 - \frac{2L\lambda}{(1-f)^2})|S_j^{t_2}|$ vertices of $S_j^{t_2}$ if $i > j$.

Take $C^{i,t_1} = \bigcap_{j \neq i, t_2=1,2,\dots,s_j} C_{i,t_1}^{j,t_2}$ for $i = 1, 2, \dots, k$, $t_1 = 1, 2, \dots, s_i$. Denote $C'_i = \bigcup_{t=1,2,\dots,s_i} C^{i,t}$ for $i = 1, 2, \dots, k$ and output (C'_1, \dots, C'_k) .

4.5.6 Algorithm *FindStrong-M-Sequence*

We will now show a technical algorithm that is fundamental for finding efficient coloring of an H -free tournament, where H is a constellation. Again as before, the naive implementation of the algorithm runs in a polynomial time. Its correctness is proven in the next section. The presented procedure is essentially a wrapper for the main algorithm *FindStrong-M-Sequence-Main* described next to it. This main algorithm operates on the smooth m -sequence that was obtained in the preprocessing performed in the initial phase of *FindStrong-M-Sequence*. The goal of the *FindStrong-M-Sequence* is to extract a strong m -sequence from an H -free tournament, where H is a given constellation. The input to the procedure is a M -big m -sequence for an appropriate parameter M .

Let us remind now some important terms regarding constellations. The *interstellar graph* of the constellation H under a star ordering θ is an undirected graph, whose vertices are the sets of leaves of the stars of H under θ and any two given vertices L_1 and L_2 are adjacent iff:

- the left point of L_1 precedes the right point of L_2 in θ and
- the left point of L_2 precedes the right point of L_1 in θ .

Take an interstellat graph of the constellation H . For each connected component C of the interstellar graph of H we denote by $Z(C)$ the union of subsets of $V(H)$ corresponding to its vertices (this is the union of some subsets of $V(H)$ of the vertices of H). We define $\mathcal{C}(Z(C))$ as follows. We say that a vertex $v \in \mathcal{C}(Z(C))$ if $v \in Z(C)$ or v is between some two vertices of $Z(C)$ under the ordering θ . Let C_1, \dots, C_k be the connected components of the interstellar graph. Note that for any given $1 \leq i < j \leq k$ either every vertex of $\mathcal{C}(Z(C_i))$ is before every vertex of $\mathcal{C}(Z(C_j))$, or every vertex of $\mathcal{C}(Z(C_j))$ is before every vertex of $\mathcal{C}(Z(C_i))$. Thus there is a natural ordering of the sets $\mathcal{C}(Z(C_i))$ for $i = 1, 2, \dots, k$ induced by the ordering of the vertices. Denote the ordered sequence of the sets $\mathcal{C}(Z(C_i))$ for $i = 1, 2, \dots, k$ as $(\mathcal{W}_1, \dots, \mathcal{W}_k)$, where a set \mathcal{W}_i is before a set \mathcal{W}_j for $1 \leq i < j \leq k$. Denote $\mathcal{W}_0 = \mathcal{W}_{k+1} = \emptyset$. For $i = 1, 2, \dots, k+1$ denote by \mathcal{R}_i the set of the vertices of H that are after all the vertices of \mathcal{W}_{i-1} and before all the vertices of \mathcal{W}_i under the ordering θ . Note that if \mathcal{R}_i is nonempty then all its elements are centers of the stars of H . Denote the set of nonempty sets \mathcal{R}_i as $\{\mathcal{M}_1, \dots, \mathcal{M}_r\}$ for some $r \geq 0$. Note that $\{\mathcal{W}_1, \dots, \mathcal{W}_k, \mathcal{M}_1, \dots, \mathcal{M}_r\}$ is a partition of the vertices of H . We denote this partition by $P_\theta(H)$. Each constellation satisfies the following: if some center of the star belongs to some $P \in P_\theta(H)$ then no leaf of this star belongs to P .

Let us explain what the algorithm *FindStrong-M-Sequence-Main* is doing. We commented on it before when we were talking about techniques used in the algorithm, but now we will be more precise. The input to the algorithm is a long m -sequence. The algorithm tries to reconstruct a constellation H in the given tournament. It wants to achieve it by selecting $|P_\theta(H)|$ transitive chunks, where θ is a constellation ordering of H , mapping each vertex of the constellation to one of the selected $|P_\theta(H)|$ chunks and looking for it in the chunk it was mapped to (if several vertices

are mapped to the same chunk then the algorithm tries to find them in different subchunks in the subdivision of the given chunk). Each transitive element of the given m -sequence is a vertex of the undirected graph G that encodes the relation between different transitive chunks. An edge between two nodes in G indicates that throughout the execution of the algorithm a particular behaviour between corresponding transitive subtournaments was detected, namely one was detected to be adjacent to the other one. This type of relation is particularly precious since both transitive chunks were previously extracted from different linear sets so there was no reason to assume before that the relation was true (it will ultimately enable us to look for different leaves of the same star in different transitive elements of the m -sequence when we will look for H later using different approach). Graph G evolves during the execution of the algorithm as new pairs of transitive chunks such that one is adjacent to the other one are detected (the evolution is conducted by adding new edges as well as replacing with new transitive chunks the old ones in the vertex set $V(G)$). The algorithm tries to reconstruct H in the tournament induced by selected $|P_\theta(H)|$ transitive chunks star by star. Whenever vertices inducing a particular star are found we say that *state 1 was reached*. Since the reconstruction of the entire H cannot succeed (input tournament is H -free) at some point, by simple analysis based on the Pigeonhole Principle, the algorithm detects two substantial transitive subchunks, such that one is adjacent to the other one. Since the initial $|P_\theta(H)|$ transitive chunks were chosen as an independent set in G , when we replace in G the original two transitive chunks by the two found transitive subchunks, we also need to add one more edge. So from the point of view of graph G in each step we are taking its independent set of size $|P_\theta(H)|$ and we add an edge between some two vertices of this set. Since the graph has 2^{h+1} nodes (and this number is the same throughout the execution of the algorithm), at every single step it has a clique or an independent set of size h . If an independent set of size h does not exist then we take an h -clique and it is easy to see that it corresponds to the strong m -sequence (every edge of the clique indicates the relation: *adjacent to* between corresponding transitive subsets). Simple calculations lead to the conclusion that the constructed strong m -sequence has desired characteristic (in terms of size of its elements, etc). If an independent set is found then new edge is added. Now notice that since whenever there exists an independent set of size h an edge is added, at some point the clique of size h will appear in G anyway. Thus we will be always able to construct a strong sequence we are looking for.

Algorithm 4.8 (*Algorithm FindStrong- M -Sequence constructing a strong m -sequence*)

- **Input:** A constellation H with $V(H) = \{1, 2, \dots, h\}$, a constellation ordering $\theta = (1, 2, \dots, h)$, an H -free tournament T and a (c, λ, ϵ) - m -sequence in T of length $k = 2^{h+2}(h+1) + 2h + 1$ which is M -big for $M \geq 2(h+2) \cdot 2^{2^{8h+2}}$. It is assumed that $\lambda \leq \frac{1}{2^{25h+6}}$.
- **Output:** An \mathcal{I} -strong $(\hat{c}, \hat{\lambda}, \epsilon)$ - m -sequence in T of length \hat{k} which is $\frac{1}{2^{24h+3}}M$ -big, where: $\hat{c} = \frac{1}{2^{24h+3}}c$, $\hat{\lambda} = 2^{2^{5h}}\lambda$, $\hat{k} = 2h^2 + 4h + 1$, $\mathcal{I} = \{h+1, 2h+1, \dots, h^2+1\}$.
- **Description:** Run algorithm *MakeSmooth* to get a smooth (c', λ', ϵ) - m -sequence (C_1, \dots, C_k) , where $c' = \frac{c}{2}$, $\lambda' = 4\lambda k$ (in algorithm *MakeSmooth* we take $f = 0$, $s_i = 1$ for $i = 1, 2, \dots, k$ and $S_i^j = C_i$). Let $\chi = 2h + 1$. Denote transitive sets: $C_{\chi+1}, C_{2\chi+2}, C_{3\chi+3}, \dots, C_{2^{h+1}\chi+2^{h+1}}$ as $T_1, \dots, T_{2^{h+1}}$ respectively. Then run algorithm *FindStrong- M -Sequence-Main* (see description below), where the arguments for *FindStrong- M -Sequence-Main* are defined as follows:
 - input m -sequence is a smooth (c', λ', ϵ) - m -sequence (C_1, \dots, C_k) computed above,
 - input graph G is an undirected graph with $V(G) = \{T_1, \dots, T_{2^{h+1}}\}$ and no edges,

- an independent set S is of the form: $S = \{T_1, \dots, T_h\}$,
- input parameter σ is the set of all stars of H ,
- input parameter ξ satisfies: $\xi = \frac{1}{2(h+2)}$,

and the coloring of vertices of the m -sequence is done as follows:

- for every T_i , $i = 1, 2, \dots, h$ color j for $j = 1, 2, \dots, h$ is assigned to the vertices of the indices: $(j-1)\lfloor \frac{|T_i|}{h+2} \rfloor + 1, \dots, j\lfloor \frac{|T_i|}{h+2} \rfloor$ in the transitive ordering of T_i (we use the convention that the first vertex in the ordering has index 1), all other vertices of T_i are colored by $h+1$,
- vertices of all other sets of the m -sequence are colored by $h+1$.

Algorithm 4.9 (*Algorithm FindStrong-M-Sequence-Main*)

- **Input:** A constellation H with $V(H) = \{1, 2, \dots, h\}$, a constellation ordering $\theta = (1, 2, \dots, h)$, an H -free tournament T and a smooth (c, λ, ϵ) - m -sequence (C_1, \dots, C_k) in T of length $k = 2^{h+2}(h+1) + 2h + 1$. Every vertex of the m -sequence is colored by a color from the set $\{1, 2, \dots, h+1\}$. For every color $j \in \{1, \dots, h+1\}$ and every $i \in \{1, \dots, k\}$ if there are vertices in C_i colored by j then at least $\xi|C_i|$ of them are colored by j . Furthermore, a nonempty subset σ of the set of stars of H is given. We are also given an undirected graph G with $V(G) = \{C_{\chi+1}, C_{2\chi+2}, C_{3\chi+3}, \dots, C_{2^{h+1}\chi+2^{h+1}}\}$, where $\chi = 2h+1$, and an independent set S of G denoted as $S = \{T'_{t_1}, \dots, T'_{t_h}\}$ for $t_1 < t_2 < \dots < t_h$.
- **Output:** An \mathcal{I} -strong m -sequence of length $\hat{k} = 2h^2 + 4h + 1$, where $\mathcal{I} = \{h+1, 2h+1, \dots, h^2 + 1\}$.
- **Description:** Take an arbitrary star $\Sigma^* \in \sigma$. Take a partitioning $P_\theta(H)$ and let assume that it is of the form: $P_\theta(H) = \{P_1, \dots, P_z\}$, where vertices of P_i are before vertices of P_j under an ordering θ for $i < j$ and z is the number of elements of the partition $P_\theta(H)$ (see: definition of $P_\theta(H)$). We will assume that Σ^* is a left star. For a right star the algorithm is completely analogous. Let n_c be such that the center of Σ^* is in P_{n_c} under θ . Denote by m_c the position that this center occupies in P_{n_c} under ordering θ (first vertex of P_{n_c} under ordering θ occupies position 1, second - position 2, etc.). Assume that Σ^* has q leaves and that all the leaves are in P_{n_l} for some n_l (note that from the definition of the constellation, n_l is the same for all leaves of Σ^* and is different than n_c). Denote by m_i for $i = 1, 2, \dots, q$ the position that i^{th} leaf occupies in P_{n_l} under ordering θ . First we check whether there exists a vertex $\rho \in T'_{t_{n_c}}$ that is colored by m_c and vertices r_1, \dots, r_q such that:

- $r_i \in T'_{t_{n_l}}$,
- r_i is colored by m_i for $i = 1, 2, \dots, q$ and
- (r_i, ρ) is a backward edge for $i = 1, \dots, q$.

As we have already noticed, since H is a constellation we know that $n_c \neq n_l$. If vertices ρ, r_1, \dots, r_q exist we say that state 1 was reached. Otherwise we say that state 0 was reached.

Assume first that state 0 was reached. But then, by Pigeonhole Principle, there exists a subset $\mathcal{C} \subseteq T'_{t_{n_c}}$ of at least $\frac{1}{q}|T'_{t_{n_c}}|$ vertices that are adjacent to all vertices of $T'_{t_{n_l}}$ colored by some fixed color $i^* \in \{1, 2, \dots, h\}$. Indeed, if state 0 was reached then we could not construct the

embedding of the left star defined above. So no matter which vertex v of $T'_{t_{n_c}}$ is taken as the center, the construction is not possible. Fix such a vertex v . We try to find leaves of the star in differently colored chunks, i.e. find a backward edge from a colored chunk to v for every color. If this is not possible then v is adjacent to all vertices for some particular color col_v . This is the place where the Pigeonhole Principle comes into action. Since altogether we have q colors, for at least $\frac{1}{q}|T'_{t_{n_c}}|$ vertices v from $T'_{t_{n_c}}$ the color col_v will be the same. In other words, at least $\frac{1}{q}|T'_{t_{n_c}}|$ vertices v from $T'_{t_{n_c}}$ will be adjacent to all vertices of $T'_{t_{n_l}}$ colored by some fixed color $i^* \in \{1, 2, \dots, h\}$.

In this scenario we replace $T'_{t_{n_c}}$ in the m -sequence by \mathcal{C} and $T'_{t_{n_l}}$ by the subset \mathcal{L} of $T'_{t_{n_l}}$ consisting of vertices colored by i^* . In the undirected graph G we replace vertex $T'_{t_{n_c}}$ by \mathcal{C} , vertex $T'_{t_{n_l}}$ by \mathcal{L} (keeping all edges of G , new vertices inherit edges adjacent to vertices that they replaced) and add an edge between vertex \mathcal{C} and vertex \mathcal{L} . Then we run on our updated m -sequence (which is not necessarily smooth) an algorithm *MakeSmooth* to make it smooth (with the same parameters as in the preprocessing phase of the algorithm *FindStrong-M-Sequence*). In G we replace all vertices by corresponding subsets extracted from them during smoothing-procedure (edges are inherited from the old graph G). Then we recolor all the vertices of the new m -sequence we obtained using the same coloring procedure that we used earlier in the algorithm *FindStrong-M-Sequence* before calling algorithm *FindStrong-M-Sequence-Main* for the first time. We replace our collection of stars by the collection of all stars of H which we call σ_H . We check whether there is a clique of size h in G . Assume first that there is not. Then, since G has 2^{h+1} vertices, it has an independent set S^* of size h . We rerun algorithm *FindStrong-M-Sequence-Main* with updated parameters $c, \lambda, \xi, \sigma, S$: $c \rightarrow \frac{c\xi}{2h}$, $\lambda \rightarrow \frac{4\lambda h^2 k}{\xi^2}$, $\xi \rightarrow \frac{1}{2(h+2)}$, $\sigma \rightarrow \sigma_H, S \rightarrow S^*$ and updated graph G . Assume now that the clique of size h was found. Then note that we can easily extract from (C_1, \dots, C_k) an \mathcal{I} -strong subsequence of length \hat{k} (this subsequence in particular contains all vertices of the clique). We output it.

It remains to consider scenario when state 1 was reached. If this is the case we remove Σ^* from σ . Let X be the set of vertices from $T'_{t_{n_c}}$ colored by m_c and let Y_i for $i = 1, 2, \dots, l$ be the set of vertices from $T'_{t_{n_l}}$ colored by m_i . Let $D_{i,j}$ be a set of vertices from $C_i \setminus (X \cup Y_1 \cup \dots \cup Y_q)$ colored by color j . Denote by $N_\rho^{D_{i,j}} \subseteq D_{i,j}$ the subset of $D_{i,j}$ consisting of vertices of $D_{i,j}$ adjacent from ρ if ρ is before all vertices of $D_{i,j}$ in the m -sequence and adjacent to ρ otherwise (the ordering in the m -sequence is induced by an ordering of sets C_i and a transitive ordering within transitive parts of the m -sequence). Similarly, denote by $N_{r_u}^{D_{i,j}} \subseteq D_{i,j}$ for $u = 1, 2, \dots, q$ the subset of $D_{i,j}$ consisting of vertices of $D_{i,j}$ adjacent from r_u if r_u is before $D_{i,j}$ in the m -sequence and adjacent to r_u otherwise. Denote $N^{D_{i,j}} = N_r^{D_{i,j}} \cap N_\rho^{D_{i,j}}$, where: $N_r^{D_{i,j}} = \bigcap_{u=1,2,\dots,q} N_{r_u}^{D_{i,j}}$. We run algorithm *MakeSmooth* on the given m -sequence, where we have: $S_i^j = N^{D_{i,j}}$, parameter f satisfies: $(1-f) = \xi(1 - \frac{\lambda h}{\xi})$ and get a new smooth m -sequence. In this new m -sequence the coloring is inherited from the old one. Now we rerun algorithm *FindStrong-M-Sequence-Main* with updated parameters c, λ, ξ : $c \rightarrow \frac{c\xi}{2}(1 - \frac{\lambda h}{\xi})$, $\lambda \rightarrow \frac{4\lambda k(h+1)}{\xi^2(1 - \frac{\lambda h}{\xi})^2}$, $\xi \rightarrow \frac{\xi}{2}(1 - \frac{\lambda h}{\xi})$, $\sigma \rightarrow \sigma \setminus \{\Sigma^*\}$.

5 Analysis of the algorithms

In this section we formally prove correctness of the algorithm that colors H -free tournaments, where H is a constellation. As a corollary we prove Theorem 2.1.

We start with some introductory observations:

5.1 Let T be a tournament. Assume that for two disjoint subsets $X, Y \subseteq V(T)$ the following holds: $d(X, Y) \geq 1 - \lambda$ for some $\lambda < 1$. Assume that $X_1 \subseteq X$, $Y_1 \subseteq Y$, $X_1 \geq c_1|X|$, $Y_1 \geq c_2|Y|$ for some $0 < c_1, c_2 < 1$. Then $d(X_1, Y_1) \geq 1 - \frac{\lambda}{c_1 c_2}$.

Proof. Let $e_{Y,X}$ be the number of directed edges from Y to X and let e_{Y_1, X_1} be the number of directed edges from Y_1 to X_1 . We have: $e_{Y,X} = (1 - d(X, Y))|X||Y| \leq \lambda|X||Y|$, since $d(X, Y) \geq 1 - \lambda$. Similarly: $e_{Y_1, X_1} = (1 - d(X_1, Y_1))|X_1||Y_1|$. Assume by contradiction that $d(X_1, Y_1) < 1 - \frac{\lambda}{c_1 c_2}$. Then, since $X_1 \geq c_1|X|$, $Y_1 \geq c_2|Y|$, we have: $e_{Y_1, X_1} > \lambda|X||Y|$. Since $e_{Y,X} \geq e_{Y_1, X_1}$, we get: $e_{Y,X} > \lambda|X||Y|$, contradiction. \blacksquare

5.2 Assume that every subtournament T_S of a tournament T contains a transitive subtournament of order at least $|T_S|^\epsilon$ for some $\epsilon > 0$. Then $\chi(T) \leq n^{1-\epsilon} \log(n)$. Besides if in every subtournament T_S of T one may find a transitive subtournament of order at least $|T_S|^\epsilon$ in time $O(h(|T_S|))$ for some nondecreasing function h , then the proper coloring of T using at most $n^{1-\epsilon} \log(n)$ colors may be constructed in time $O(n^{1-\epsilon} \log(n)h(n) + n^2 \log(n))$.

Proof. In the preprocessing phase we sort each adjacency list. This requires $O(n^2 \log(n))$ time. Find a transitive subtournament T_1 of T with $|T_1| \geq (\frac{n}{2})^\epsilon$ and delete it from T . To perform a deletion we first sort the vertices of the found tournament and this can be done in $O(n \log(n))$ time. Then we get rid of all the adjacency lists that are related to the vertices from the found tournament. This can be done in $O(n \log(n))$ time simply by going through each adjacency list and performing a binary search in the sorted sequence of the vertices from the transitive subtournament. Finally we delete vertices of the transitive subtournament from all remaining adjacency lists and this can be done in $O(n|T_1| \log(n))$ time. We keep finding transitive subtournaments of order at least $(\frac{n}{2})^\epsilon$ as long there are at least $\frac{n}{2}$ vertices in the tournament. The total time spent for running this subprocedure is: $O(h(n)n^{1-\epsilon} + n \log(n) + n|T_1| \log(n))$. When we reach the state with less than $\frac{n}{2}$ vertices remaining, we have found $O(n^{1-\epsilon})$ transitive subtournaments: T_1, T_2, \dots . We then apply the same subprocedure on the remaining graph of less than $\frac{n}{2}$ vertices. We stop when there are no vertices left and by that time we have partitioned tournament T into transitive subtournaments. If we denote by $H(n)$ the number of the transitive subtournaments found then we have the following simple recurrence formula: $H(n) \leq (\frac{n}{2})^{1-\epsilon} + H(\frac{n}{2})$, which immediately gives us: $H(n) \leq n^{1-\epsilon} \log(n)$. By coloring each transitive tournament with the same color and using different colors for different transitive subtournaments we get a proper coloring of T that uses at most $n^{1-\epsilon} \log(n)$ colors. If we denote by $T(n)$ the total running time of the algorithm then the above observations (and simple calculations) give us the following formula: $T(n) = O(n^{1-\epsilon} \log(n)h(n) + n^2 \log(n))$. That completes the proof. \blacksquare

The following theorem turns out to be very important to prove the correctness of our coloring algorithm. It also explains how a polynomial lowe bound on the size of the transitive subtournament can be obtained.

5.3 Let be T be a tournament and let $A_1, T_1 \subseteq V(T)$. Denote by T^{A_1} a tournament induced by A_1 . Assume that (A_1, T_1) is (c, ϵ_c) -saturated, $c > 0$, and T^{A_1} is ϵ_c -transitive, where $\epsilon_c = \frac{\log(1-c)}{\log(c)}$. Assume furthermore that a transitive subtournament of T^{A_1} of size at least $|A_1|^{\epsilon_c}$ might be found in time $h(|A_1|)$ for some function h . Then T is ϵ_c -transitive and its transitive subtournament of size at least $|T|^{\epsilon_c}$ might be found in time $h(|A_1|) + t_m$, where t_m is the size of the largest transitive subtournament of T .

Proof. Denote $n = |T|$. Let T' be a transitive subtournament of order at least $|A_1|^{\epsilon_c}$, found in T^{A_1} . Note that if we merge it with a tournament induced by T_1 then we get a transitive subtournament. It only suffices to prove now that this bigger transitive subtournament, denote it as T^l , satisfies: $|T^l| \geq n^{\epsilon_c}$. We have: $|T^l| \geq (cn)^{\epsilon_c} + cn^{\epsilon_c}$, since $|A_1| \geq cn$. Since $\epsilon_c = \frac{\log(1-c)}{\log(c)}$, we obtain: $|T^l| \geq n^{\epsilon_c}$. That completes the proof. \blacksquare

Now we prove correctness of the algorithm 4.4 and analyze its running time. Let us denote: $\lambda_i = (\frac{\lambda^2}{4^k h^{4k}})^{2^i h^{2i}}$ for $i = 0, \dots, k$.

5.4 *If $|T| \geq \frac{2^{k+1}(h+1)h^{2k}}{\lambda_k^{hk}}$ then algorithm 4.4 constructs a (c, λ) - l -sequence for $c = \frac{\lambda^{hk}}{2^k h^{2k}}$, where 2^k is the length of the sequence and h is the size of the forbidden subtournament, and runs in polynomial time.*

Proof. Note first that $\lambda_i \leq \frac{\lambda^{2h^i}}{4^i h^{4i}} \lambda$ for $i = 1, 2, \dots$. We call this property of the sequence $\{\lambda_i\}$ for $i = 0, 1, \dots$ the α -property. The algorithm trivially works for $k = 0$ so we can assume from now on that $k > 0$. Let $h = |H|$ and $n = |T|$, where H is a forbidden tournament. Note that the algorithm stops when the subroutine 4.1 is called with $h^r = 1$ or the algorithm 4.4 itself is called with $k = 0$. Note also that if the former holds then the last h calls in the recursive call-tree on the path ending at that call were the calls of the subroutine 4.1. But then we can take last h vertices v^* found in h last calls of the subroutine 4.1 and they induce a copy of H in T , contradiction. Let us explain in detail why this is the case. Take the first vertex v^* from the sequence of h consecutive ones and call it v_1^* . Notice that the remaining $h - 1$ consecutive calls will operate on the set of sets $\{N_j(v_1^*)\}$. The remaining $h - 1$ vertices v^* induce a copy of $H \setminus \{v_1^*\}$ in such a way that in the embedding each vertex of $H \setminus \{v_1^*\}$ resides in the different set $N_j(v_1^*)$. But then we can take this embedding of $H \setminus \{v_1^*\}$, add vertex v_1^* and from the definition of the sets $N_j(v_1^*)$ we conclude that the constructed set of h vertices induces a copy of H . Thus if the algorithm stops then algorithm 4.4 is recursively called with $k = 0$. Note also that the algorithm must stop since in the recursive call-tree there does not exist a path of h calls of the subroutine 4.1 (from what we have said so far) and whenever algorithm 4.4 is called parameter k is being decreased by 1. Note that, since $n \geq \frac{2^{k+1}(h+1)h^{2k}}{\lambda_k^{hk}}$, whenever algorithm 4.4 is recursively called, the H -free tournament it operates on is of size at least $2(h+1)$. To see this, consider one call of the algorithm 4.4, and let i be its second parameter. Note first that after one call of algorithm 4.4 and at most h consecutive calls of subroutine 4.1 the H -free tournament T^2 which is the last parameter of the next call of algorithm 4.4 is of size at least $(\frac{|T^1|}{h+1} - 1)\lambda_i^h \frac{1}{h-1} = |T^1|(\frac{1}{h+1} - \frac{1}{|T^1|})\frac{\lambda^h}{h-1}$, where T^1 is the last parameter of the previous call of algorithm 4.4. Finally, note that on the path of the tree of recursive calls there are at most k consecutive calls of algorithm 4.4 and that $\lambda_k \leq \lambda_i$ for $i = 0, 1, \dots, k-1$. Assume now that in the subroutine 4.1 we reached the state when the two sets $W, S_{t_{j^*}}$ were found (see: description of the algorithm from the previous section). Assume without loss of generality that $d(W, S_{t_{j^*}}) \geq 1 - \lambda_i$. Denote by n_0 the size of the tournament T^r which is the parameter of the last call of algorithm 4.4 preceding in the call-tree the construction of W and $S_{t_{j^*}}$. Note that inductively sequences $(A_1, \dots, A_{2^{k-1}})$ and $(A'_1, \dots, A'_{2^{k-1}})$ are both $(c(H, k-1, \lambda), \lambda)$ - l -sequences. Note also that we have: $|W|, |S_{t_{j^*}}| \geq s$, where $s = \lfloor \frac{n_0}{h+1} \rfloor \lambda_k^i \frac{1}{h-1}$ (this comes from the previous observation that between two consecutive runs of algorithm 4.4 we have at most h recursive runs of the subroutine 4.1). We have: $s \geq (\frac{n_0}{h+1} - 1)\lambda_i^h \frac{1}{h-1}$, thus $s \geq n_0(\frac{1}{h+1} - \frac{1}{n_0})\lambda_i^h \frac{1}{h-1}$. Since at any point of the execution of the algorithm an H -free tournament we are dealing with has size at least $2(h+1)$, we can conclude that $|W|, |S_{t_{j^*}}| \geq fn_0$, where $f = \frac{\lambda^h}{2(h-1)}$. Then, since $\lambda_i \leq \lambda c^2(H, i-1, \lambda)$ (which

follows from the α -property) and $c(H, i, \lambda) \leq fc(H, i - 1, \lambda)$, using Theorem 5.1, we can deduce that the sequence $(A_1, \dots, A_{2^{k-1}}, A'_1, \dots, A'_{2^{k-1}})$ is a $(c(H, k, \lambda), \lambda)$ - l -sequence.

To prove that algorithm 4.4 runs in polynomial time, note that time spent by the algorithm between two its recursive consecutive runs on the path of recursive calls is polynomial. Therefore, if $T(k)$ denotes time spent by the algorithm to find a l -sequence of length 2^k , then we have: $T(k) \leq \text{poly}(n) + 2T(k - 1)$. Since k is a constant, $T(k)$ is clearly polynomial in n . \blacksquare

Now we prove correctness of the algorithm 4.5.

5.5 Algorithm 4.5 computes a clique $\{v_1, \dots, v_k\}$.

Proof. Assume first that at each stage of the algorithm sets N_i^v are nonempty. We will prove it later. Note that from Theorem 5.1 we know that $|W_i| \leq \frac{|V_1|}{2^k}$. Thus we have $|W_2 \cup \dots \cup W_k| \leq |W_2| \cup \dots \cup |W_k| \leq k \frac{|V_1|}{2^k} \leq \frac{|V_1|}{2}$. Thus a set $V_1 \setminus (W_2 \cup \dots \cup W_k)$ is nonempty so we can always find v_1 . It is obvious that if we combine the clique found in the next call of the algorithm 4.5 with vertex v_1 then we get a clique. Note also that since $|N_i^v| \geq (1 - 2k\lambda)|V_i|$, by Theorem 5.1, we have: $d(V'_i, V'_j) \geq 1 - \frac{\lambda}{(1-2k\lambda)^2}$ for $2 \leq i < j \leq k$, so we update the parameters of the algorithm correctly. Notice that the first run of the algorithm 4.5 is for $\lambda \leq \frac{1}{k3^{2k+1}}$ and that altogether there are exactly k calls of the algorithm, where in each call we update: $\lambda \rightarrow \frac{\lambda}{(1-2k\lambda)^2}$. Thus each time the algorithm 4.5 is called we have: $\lambda \leq \frac{1}{3k}$. Therefore in particular, whenever sets N_i^v are calculated they are always nonempty. That completes the proof of the correctness of the algorithm 4.5. \blacksquare

Now we prove correctness of the algorithm 4.6 and analyze its running time.

5.6 Algorithm 4.6 computes a $(\min(c, (\frac{c}{2})^\epsilon), \Lambda, \epsilon)$ - m -sequence which is $(\log(cn) - 2)$ -big and runs in time $O(h(A^m)n^{1-\epsilon}) + \text{poly}(n)$, where $\text{poly}(n)$ is a polynomial factor.

Proof. Denote $n = |T|$. Denote $A_{2i}^p = T_1^i \cup \dots \cup T_{r_i}^i$ for $i = 1, 2, \dots, k$. Note that $|A_{2i}^p| \geq \frac{|A_{2i}|}{2}$ for $i = 1, 2, \dots, k$. Denote $A'_{2i+1} = A_{2i+1}$ and $A'_{2i} = A_{2i}^p$ for $i = 1, 2, \dots, k$. Then, using Theorem 5.1, we can deduce that $d(A'_i, A'_j) \geq 1 - 4\lambda$ for $1 \leq i < j \leq 2k + 1$. Indeed, $|A'_i| \geq \frac{|A_i|}{2}$ and $|A'_j| \geq \frac{|A_j|}{2}$ and furthermore: $d(A_i, A_j) \geq 1 - \lambda$. Now take some A'_i and A'_j for $1 \leq i < j \leq 2k + 1$ and corresponding sets V_i and V_j in G . Note that since $d(A'_i, A'_j) \geq 1 - 4\lambda$, using Theorem 5.1, we can conclude that the number of edges going between V_i and V_j is at least $(1 - \lambda_0)|V_i||V_j|$, where $\lambda_0 = \frac{1}{(2k+1)3^{4k+3}}$. Let us explain in detail why this is the case. Assume otherwise. Then there are at least $\lambda_0|V_i||V_j|$ pairs of elements (x, y) from V_i and V_j that are not adjacent in G (notice that each x and y is a subset). Each element from x has the same size and each element y has the same size (even though the size of x does not have to be the same as the size of y). Denote the size of each element of V_i by s_1 and the size of each element of V_j by s_2 . Then we can conclude that there are more than $\lambda_0|V_i||V_j|s_1s_2\Lambda$ edges going from A'_j to A'_i (since by definition of the nonedge of G , each nonedge (x, y) introduces at least $\Lambda|x||y|$ edges from A'_j to A'_i). Notice that the size of A'_i is $|V_1|s_1$ and the size of A'_j is $|V_2|s_2$. Thus there are more than $\lambda_0\Lambda|A'_i||A'_j|$ edges going from A'_j to A'_i . On the other hand, since $d(A'_i, A'_j) \geq 1 - 4\lambda$, we know that the number of edges going from A'_j to A'_i is at most $4\lambda|A'_i||A'_j| \leq \lambda_0\Lambda|A'_i||A'_j|$, where the last inequality follows from the fact that (by assumptions of the theorem) $\lambda \leq \lambda_0\frac{\Lambda}{4}$. We get a contradiction. Thus indeed the number of edges going between V_i and V_j is at least $(1 - \lambda_0)|V_i||V_j|$. But then we see that all conditions necessary to run algorithm 4.5 are satisfied. The parameter k in the statement of the algorithm 4.5 correspond to $2k + 1$ in our

setting since our m -sequence is of length $2k+1$. Thus λ_0 from our setting corresponds to the upper bound on λ from the statement of algorithm 4.5.

Note also that a clique found by this algorithm in G corresponds to the (c', Λ, ϵ) - m -sequence for $c' = \min(c, (\frac{c}{2})^\epsilon)$. This comes from the fact that each T_j^i satisfies $|T_j^i| \geq (\frac{c}{2})^\epsilon n^\epsilon$. This m -sequence is $(\log(cn) - 2)$ -big since each extracted T_j^i satisfied: $|T_j^i| \geq \log(cn) - 2$. This is the case since tournaments T_j^i are extracted from tournaments of size at least $s \geq \frac{cn}{2}$ and by classic Ramsey argument, each such tournament has a transitive subtournament of order at least $\log(s) - 1$.

Ley us analyze now the running time of the algorithm. Note first that if W is an N -vertex tournament then its transitive subtournament of size at least $\log(N) - 1$ may be found as follows: Take an arbitrary vertex $v_1 \in V(W)$. Let $N_{v_1}^-$ be the set of its inneighbors in W and let $N_{v_1}^+$ be the set of its outneighbors in W . If $N_{v_1}^- \geq \frac{N-1}{2}$ then let $W_1 = W|N_{v_1}^-$, otherwise let $W_1 = W|N_{v_1}^+$. Now consider a tournament W_1 and repeat the procedure by taking an arbitrary vertex $v_2 \in V(W_1)$ and considering sets of its inneighbors and outneighbors in W_1 , etc. Using this procedure we get the sequence of vertices: v_1, \dots, v_r for some r . It is easy to see that $r \geq \log(N) - 1$ and that $\{v_1, \dots, v_r\}$ is a transitive subset. Trivial implementation of this algorithm clearly runs in $\text{poly}(N)$ time. We use the procedure we have just described to find transitive subtournament of size at least $\log(\frac{cn}{2}) - 1$ in algorithm 4.6. Note that the only possibly nonpolynomial part of the running time corresponds to extracting transitive subtournaments with the use of procedure h . Fix some A_i . Each transitive subset extracted from A_i is of size at least $(\frac{|A_i|}{2})^\epsilon$. Thus the number of extracted transitive subsets from any given A_i is $O(|A_i|^{1-\epsilon}) = O(n^{1-\epsilon})$. Extracting a transitive subset requires time $O(h(|A_i|)) = O(h(A^m))$. Finally note that we have a fixed number of sets A_i . That completes the analysis of the running time of the algorithm. \blacksquare

Now we prove correctness of the algorithm 4.7.

5.7 Algorithm 4.7 computes a smooth (c', λ', ϵ) - m -sequence, a smooth (c', λ', ϵ) - t -sequence or a smooth (c', λ') - l -sequence (C'_1, \dots, C'_k) respectively, where $c' = \frac{c}{2}(1-f)$, $\lambda' = \frac{4\lambda L}{(1-f)^2}$, $L = s_1 + \dots + s_k$ and $C'_i \subseteq \bigcup_{j=1,2,\dots,s_i} S_i^j$ for $i = 1, 2, \dots, k$. Besides we have: $|C'_i \cap S_i^j| \geq \frac{cn}{2}(1-f)$ for $i = 1, 2, \dots, k$, $j = 1, 2, \dots, s_i$.

Proof. We assume without loss of generality that a (c, λ, ϵ) - m -sequence is given in the input. Let T be a tournament with the (c, λ, ϵ) - m -sequence and denote $n = |T|$. Note first that from Theorem 5.1 we know that $d(S_i^{t_1}, S_j^{t_2}) \geq 1 - \frac{\lambda}{(1-f)^2}$. Using Theorem 5.1 again we can conclude that $|S_i^{t_1} \setminus C_{i,t_1}^{j,t_2}| \leq \frac{|S_i^{t_1}|}{2L}$. To see that assume without loss of generality that $i < j$ (for $i > j$ the analysis is exactly the same). Note first that $d(S_i^{t_1}, S_j^{t_2}) \geq 1 - \frac{\lambda}{(1-f)^2}$. Thus the number of directed edges from $S_j^{t_2}$ to $S_i^{t_1}$ is at most $\frac{\lambda}{(1-f)^2} |S_i^{t_1}| |S_j^{t_2}|$. However if $|S_i^{t_1} \setminus C_{i,t_1}^{j,t_2}| > \frac{|S_i^{t_1}|}{2L}$ then the number of vertices of $S_i^{t_1}$ that are adjacent to less than $(1 - \frac{2L\lambda}{(1-f)^2}) |S_j^{t_2}|$ vertices of $S_j^{t_2}$ is more than $\frac{|S_i^{t_1}|}{2L}$ which clearly indicates that the number of directed edges from $S_j^{t_2}$ to $S_i^{t_1}$ is more than $\frac{2L\lambda}{(1-f)^2} |S_j^{t_2}| |S_i^{t_1}|$. This is a contradiction according to what we have noted before.

Therefore we have: $|C^{i,t_1}| = |S_i^{t_1} \setminus \bigcup_{j \neq i, t_2=1,\dots,s_j} (S_i^{t_1} \setminus C_{i,t_1}^{j,t_2})| \geq |S_i^{t_1}| - L \frac{|S_i^{t_1}|}{2L} \geq \frac{|S_i^{t_1}|}{2}$ (notice that the number of terms in the sum from the last sequence of inequalities is at most L). Now take a vertex $v \in C^{i,t_1}$ and take some $j > i$, $t_2 \in \{1, 2, \dots, s_j\}$. Note that there are at least $(1 - \frac{2L\lambda}{(1-f)^2}) |S_j^{t_2}|$ vertices in $|S_j^{t_2}|$ adjacent from v . Since $|C^{j,t_2}| \geq \frac{|S_j^{t_2}|}{2}$, we can conclude that $d(\{v\}, C^{j,t_2}) \geq 1 - \frac{\frac{2L\lambda}{(1-f)^2} |S_j^{t_2}|}{\frac{|S_j^{t_2}|}{2}}$. Thus $d(\{v\}, C^{j,t_2}) \geq 1 - \frac{4\lambda L}{(1-f)^2}$. Similar analysis can be done for

$j < i$, $t_2 \in \{1, 2, \dots, j\}$. Now note that $|C'_i \cap S_i^j| \geq \frac{|S_i^j|}{2} \geq \frac{(1-f)cn}{2}$. That completes the proof. \blacksquare

Now we prove correctness of the algorithm 4.8.

5.8 *Algorithm 4.8 computes an \mathcal{I} -strong $(\hat{c}, \hat{\lambda}, \epsilon)$ - m -sequence of length k' which is $\frac{1}{2^{2h+3}}M$ -big, where $\hat{c} = \frac{1}{2^{2h+3}}c$, $\hat{\lambda} = 2^{2^{5h}}\lambda$, $\mathcal{I} = \{h+1, 2h+1, \dots, h^2+1\}$ and $k' = 2h^2 + 4h + 1$, assuming that $\lambda \leq \frac{1}{2^{25h+6}}$ and $M \geq 2(h+2) \cdot 2^{2^{8h+2}}$.*

Proof. Note that from the definition of a graph G and the way it is updated it is clear that when the algorithm stops it returns an \mathcal{I} -strong m -sequence. Indeed, every clique $\{C_{i_1}, C_{i_2}, \dots, C_{i_h}\}$ in G satisfies: $d(C_{i_{j_1}}, C_{i_{j_2}}) = 1$ for $1 \leq j_1 < j_2 \leq h$. Now note that state 0 may be achieved throughout the execution of the algorithm at most $\frac{|V(G)|^2}{2}(1 - \frac{1}{h-1}) + 1$ times, where $|V(G)| = 2^{h+1}$. This is true since when state 0 is achieved a new edge is added to G . Graph G cannot have more than $\frac{|V(G)|^2}{2}(1 - \frac{1}{h-1}) + 1$ edges since, by Turan's Theorem, if G has $\frac{|V(G)|^2}{2}(1 - \frac{1}{h-1}) + 1$ edges then it has a clique of size h . Note also that state 1 may be achieved at most $h-1$ times in a row since otherwise we can merge stars found at each of the consecutive stages when state 1 was achieved to reconstruct H . That contradicts the fact that T is H -free. We can conclude that the algorithm stops. It only suffices to show that all its parameters are correctly updated. This comes directly from algorithm 4.7 that was analyzed before, Theorem 5.1 and the fact that at every stage of the algorithm each transitive set of the m -sequence has at least $2(h+2)$ vertices (thus every expression of the form $\lfloor \frac{|T|}{h+1} \rfloor$, where T is a transitive part of the m -sequence, may be bounded from below by $\frac{|T|}{2(h+1)}$). We call this last property an Ω -property and will prove it later. Knowing that, we are ready to analyze in more detail the updates when states: 0 and 1 are reached. To see that whenever state 0 is reached all parameters are correctly updated, notice that while replacing elements of the m -sequence we decrease the size of each set of the sequence at most $\frac{\xi}{h}$ times. Now assume that state 1 was reached. Note that each $D_{i,j}$ satisfies: $|D_{i,j}| \geq \xi |C_i|$. Since the m -sequence is smooth, by Theorem 5.1, we have: $d(\rho, D_{i,j}) \geq 1 - \frac{\lambda}{\xi}$. Thus we have $|N_\rho^{D_{i,j}}| \geq c\xi(1 - \frac{\lambda}{\xi})$. Similarly: $|N_{r_i}^D| \geq c\xi(1 - \frac{\lambda}{\xi})$ for $i = 1, \dots, q$. Therefore we have: $|N^{D_{i,j}}| \geq c\xi(1 - \frac{\lambda h}{\xi})$. But then we can run algorithm 4.7 with $(1-f) = \xi(1 - \frac{\lambda h}{\xi})$. That observations enables us to finish the analysis of the parameters' updates when state 1 is reached.

It remains to prove an Ω -property. The fact that at every stage of the algorithm each transitive set of the m -sequence has at least $2(h+2)$ vertices is implied by our next remark. One can notice that under our choice of the initial values of parameters c, λ we have: $\frac{\lambda h}{\xi} \leq \frac{1}{2}$ and $\xi \geq \frac{1}{2^{2h-1}(h+2)}$ at every stage of the algorithm. Note that when state 0 is reached and we update the parameters, we have: $\xi = \frac{1}{2(h+2)}$. On the other hand, since $\frac{\lambda h}{\xi} \leq \frac{1}{2}$, when stage 1 is reached and we update the parameters, we have: $\xi_{new} \geq \frac{\xi_{old}}{4}$, where ξ_{new} is the value of ξ after the update and ξ_{old} is the one before the update. Thus, since state 1 may be achieved at most $h-1$ times in a row, we get: $\xi \geq \frac{1}{2^{2h-1}(h+2)}$ at every stage of the execution of the algorithm. Now, from what we have said so far, we can conclude that whenever state 0 is achieved we have: $\lambda_{new} \leq 4h^2k(h+2)^22^{4h-2}\lambda_{old}$ and whenever state 1 is achieved we have: $\lambda_{new} \leq 16k(h+1)(h+2)^22^{4h-2}\lambda_{old}$, where: λ_{new} is the value of λ after the update and λ_{old} is the one before the update. Thus at every stage of the algorithm we also have: $\lambda \leq \frac{1}{2^{2h}(h+2)}\lambda_{init}$, where λ_{init} is value of the parameter λ at the very beginning of the algorithm. Whenever state 0 is achieved we also have: $c_{new} \geq \frac{c_{old}}{2^{2h}h(h+2)}$ and whenever state 1 is achieved we have: $c_{new} \geq \frac{c_{old}}{2^{2h+1}(h+2)}$, where: c_{new} is the value of c after the update and c_{old} is the one before the update. Now note that $k \leq 2^{2h+3}$. Notice that from what we have said before

we know that state 0 is achieved at most 2^{2h+2} times during execution of the algorithm 4.8 and state 1 is achieved at most $(h-1)2^{2h+2}$ times. All these observations and some simple calculations imply that at every stage of the algorithm each transitive set of the m -sequence is indeed of size at least $2(h+2)$. Therefore Ω -property is satisfied.

Thus an \mathcal{I} -strong m -sequence output by the algorithm is a $(\hat{c}, \hat{\lambda}, \epsilon)$ - m -sequence which is $\hat{c}M$ -big for parameters $\hat{c}, \hat{\lambda}$ defined in the algorithm. \blacksquare

Now we prove correctness of the algorithm 4.1.

5.9 *Assume that we are given a m -sequence of the H -free tournament T with $h = |H|$. Assume furthermore that this sequence is an \mathcal{I} -strong (c, λ, ϵ) - m -sequence of length $k = 2h^2 + 4h + 1$, where $\lambda \leq \frac{1}{2^{25h^2}h}$, $n = |T| \geq \frac{2^{21h^2}}{c}$, $M \geq 2^{21h^2}$, $\epsilon = \frac{\log(1-\hat{c})}{\log(\hat{c})}$ and $\hat{c} = \frac{c}{2^{7h^2}}$. Then algorithm 4.1 computes a transitive subtournament of T of order at least $|T|^\epsilon$.*

Proof. Our analysis is very similar to the one conducted in the proof of the correctness of the algorithm 4.8. Note that it cannot be the case that during the execution of the algorithm all the stars of H were found and σ was empty at some point of the execution since after combining these stars one can reconstruct H in T which contradicts the fact that T is H -free. Note also that after our choice of initial values of the parameter λ during the entire execution we have: $\frac{\lambda h}{\xi} \leq \frac{1}{2}$ and at every point of the execution of the algorithm each set of the m -sequence under consideration is of size at least 6 (this follows by simple but tedious calculations, similar to those presented in the analysis of algorithm 4.8). Therefore now we can repeat analysis of the algorithm 4.8. Thus we will not discuss details again. Note only that during the entire execution of the algorithm we have: $\xi \geq \frac{1}{4^{h+2}}$. This is true since whenever we update parameter ξ we have: $\xi_{new} \geq \frac{\xi_{old}}{4}$, where ξ_{new} is a value of ξ after the update and ξ_{old} is the one before the update. Similarly, whenever we update parameter λ we have: $\lambda_{new} \leq 4^{2h+6}k\lambda_{old}$, where λ_{new} is a value of λ after the update and λ_{old} is the one before the update. And finally, whenever we update parameter c we have: $c_{new} \geq \frac{c_{old}}{4^{h+2}}$, where c_{new} is a value of c after the update and c_{old} is the one before the update. Thus at every point of the execution of the algorithm we have: $c \geq \frac{c_b}{2^{7h^2}}$, where: c_b is the value of the parameter c at the very beginning of the algorithm. The only new part that we will focus on concerns running procedure \mathcal{P} . Note that after running it we obtain an $(\frac{c}{2^{7h^2}}, \epsilon)$ -saturated pair. Thus it suffices to note that, because of Theorem 5.3, under our choice of ϵ , constructed transitive subtournament is of order at least $|T|^\epsilon$. \blacksquare

Let us prove now that the algorithm 4.3 is correct. We will also prove Theorem 2.1.

5.10 *Let H be a constellation of order h and let T be an H -free tournament. Then the algorithm 4.3 outputs a proper coloring of the vertices of H that uses at most $|T|^{1-\frac{1}{2^{250h^2+1}}}\log(|T|)$ colors. Besides the first color class it finds is a transitive tournament of order at least $|T|^{\frac{1}{2^{250h^2+1}}}$.*

Proof. Correctness of the algorithm is a simple consequence of the fact that algorithms: 4.4, 4.6, 4.8 and 4.1 are correct. In particular, under our initial choice of parameter λ we have: $c = \frac{1}{2^{250h^2}}$ during execution of the algorithm 4.1. Thus we can take as an ϵ every positive value no greater than $\frac{\log(1-c)}{\log(c)}$. Note that in order to use algorithms: 4.4, 4.6, 4.8 and 4.1 the conditions on the sizes of the elements of m/l -sequences given as an input need to be satisfied. One can check that all those conditions are satisfied whenever a tournament T we proceed with is of size at least: $2^{2^{2^{10h^2+30h}}}$. And one can also easily note that for $\epsilon_1 = \frac{1}{2^{250h^2+1}}$ we trivially have: $N^{\epsilon_1} < 2$ for $n < 2^{2^{2^{10h^2+30h}}}$.

Thus every n -vertex tournament T with $n = |T| < 2^{2^{10h^2+30h}}$ contains a transitive subtournament of size at least $|T|^{\epsilon_1}$. Therefore the EH coefficient of a constellation H is at least $\frac{1}{2^{50h^2+1}}$. That, because of Theorem 5.2, completes the proof of the correctness of the algorithm 4.3. Let us analyze the running time of the algorithm. The only, possibly nonpolynomial factor comes from the execution of the algorithm 4.6 and from the execution of the algorithm 4.1 (to be more precise: from extracting transitive subtournaments by procedures: Sub and \mathcal{P}). Thus let us take advantage of the analysis of the running time of the algorithm 4.6 and the algorithm 4.1. Denote by $T(n)$ the running time of the algorithm 4.1. We have the following straightforward recursive formula: $T(n) \leq O(T((1-c)n)n^{1-\epsilon}) + poly(n)$, where $poly(n)$ is a polynomial factor. Similarly, if $S(n)$ is the running time of the algorithm 4.6, then we have: $S(n) \leq O(S((1-c)n)n^{1-\epsilon}) + poly(n)$. If we denote by $G(n)$ the running time of the algorithm 4.3 then we have: $G(n) = O(n(T(n) + S(n))) + poly(n)$. Thus one can check that for $C > 0$ large enough function $e^{C\log(n)^2}$ is an upper bound for $T(n)$, $S(n)$ and $G(n)$. \blacksquare

6 Further remarks

In this section we discuss some applications of the techniques presented in the paper.

6.1 The Erdős-Hajnal conjecture for constellations

The presented algorithm gives a constructive proof of the theorem stating that every constellation satisfies the Erdős-Hajnal conjecture. The theorem was first proven by [4]. However that proof was not constructive. Besides, because it used the regularity lemma, it gave much weaker lower bounds on the EH coefficients of constellations. Our bound is also very small but needless to say, it is much bigger than the best lower bound that can be obtained with an approach that uses the regularity lemma. We prove that the EH coefficient of a constellation H is at least $\frac{1}{2^{50h^2+1}}$, where $h = |H|$.

6.2 The Erdős-Hajnal conjecture for nonprime tournaments

We start with one more useful notation. For any tournament H with vertex set $V(H) = \{v_1, \dots, v_h\}$ and for any tournaments F_1, \dots, F_h let $H(F_1, \dots, F_h)$ denote the tournament obtained from H by replacing each v_i with a copy of F_i , and making a vertex of the copy of F_i outadjacent to a vertex of a copy of F_j , $j \neq i$, if and only if $(v_i, v_j) \in E(H)$. The copies of F_i , $i = 1, \dots, h$, are assumed to be vertex disjoint.

Let H, F be tournaments satisfying the Erdős-Hajnal conjecture with some $\epsilon(H), \epsilon(F) > 0$. Let $V(H) = \{v_1, \dots, v_h\}$. Denote by $H(F, v_2, \dots, v_h)$ the tournament obtained from H by replacing v_1 by F . We say that a tournament $H(F, v_2, \dots, v_h)$ was obtained from H and F by a *substitution procedure* (we substitute v_1 with F). Note that if $|F| > 1$ and $h > 1$ then $H(F, v_2, \dots, v_h)$ is not prime since $V(F)$ is a nontrivial homogeneous set. It was proven in [1] that tournament $H(F, v_2, \dots, v_h)$ also satisfies the conjecture with $\epsilon(H(F, v_2, \dots, v_h)) = \delta\epsilon(H)$ for every $\delta < \frac{\epsilon(F)}{\epsilon(H)+h\epsilon(F)}$. To be more precise, in [1] the analogous result for undirected graphs was proven and the proof of the directed version was not given explicitly. However a proof of the directed version is completely analogous to the one for the undirected version - cliques/stable sets are replaced by transitive subsets and induced subgraphs by subtournaments. Let \mathcal{F} be some family of tournaments for which the conjecture is known and let $\hat{\mathcal{F}}$ be the closure of \mathcal{F} under taking substitutions. Then we can conclude that every member of $\hat{\mathcal{F}}$ also satisfies the conjecture. Besides, if we can construct a polynomial-size transitive

subtournament for every member of \mathcal{F} then we can also construct an algorithm that can do the same for every member of $\hat{\mathcal{F}}$. It is so since the proof that substitutions preserve the Erdős-Hajnal property, as presented in [1], is constructive. Thus as a corollary of the algorithm presented in the previous section we obtain algorithms for coloring H -free tournaments with $O(n^{1-\epsilon} \log(n))$ colors, where H is taken from the closure $\hat{\mathcal{C}}$ of the family of constellations \mathcal{C} . In fact, techniques used in the algorithm from the previous section may also be used for some tournaments that are not constellations (see: the next subsection) thus we obtain coloring algorithms for even larger classes of tournaments. We should note here that a straightforward algorithmic version of the proof that the substitution procedure preserves the Erdős-Hajnal conjecture has a sub-exponential running time $e^{\Omega(n^k)}$ for some constant $0 < k < 1$ since it needs to examine all subsets of size $\lceil n^\delta \rceil$ of the n -element set (see: [1], pages: 4-5). Therefore whenever we use an algorithmic version of the substitution procedure we get a coloring algorithm that runs in the sub-exponential time. Since the algorithm presented in the previous section required only quasi-polynomial time, the question arises whether it is possible to get an algorithmic version of the proof of the substitution procedure that also requires only quasi-polynomial time. It seems that the method used in the proof proposed in [1] cannot be easily modified to improve the running time. However a completely different proof may potentially have this property. Interestingly, the algorithm proposed in the previous section to color H -free tournaments, where H is a constellation, uses different techniques from those that were used in [1] to prove the mentioned property of the substitution procedure and that enabled us to obtain quasi-polynomial running time. An open question is whether this running time may be improved to polynomial.

6.3 The Erdős-Hajnal conjecture for small tournaments

It turns out that many tournaments may be obtained from constellations by the substitution procedure. In particular, this is true for all tournaments on 5 vertices except for the tournament C_5 (see: Theorem 6.1). Thus, according to what we have said before, for all tournaments H on at most 5 vertices except the tournament C_5 we get an algorithm running on an arbitrary H -free tournament, finding its polynomial-size subtournament and coloring requiring only $O(n^{1-\epsilon} \log(n))$ colors. At the same time we get a constructive proof of the Erdős-Hajnal conjecture for those tournaments. In fact we can say even more. It is true (though we will not show it here) that a similar method that was used in the algorithm presented in this paper may be used along with the methods presented in [5] to give a quasi-polynomial time algorithm that colors every n -vertex C_5 -free tournament with $O(n^{1-\epsilon(C_5)} \log(n))$, where $\epsilon(C_5)$ can be exactly calculated and given in closed-form (again, the regularity lemma is not required).

We do not present that algorithm in this paper because of length constraints. The idea behind the proof is that we can construct an arbitrary m -sequence in the same algorithmic way as we did in this paper for constellations. This is true since the construction of the m -sequence does not use the specific structure of the constellation. The only thing we need to know is that a tournament is defined by a forbidden pattern H . When we have the m -sequence we try to reconstruct C_5 using one specific ordering of its vertices under which the graph of backward edges is a tree. Since the input tournament is C_5 -free we wont be able to succeed. Then we show that we either get a linear set exactly adjacent to/from the big transitive chunk (as in the constellation proof) and that by induction immediately leads to the explicit bound on the size of the transitive subtournament or we obtain another graph of backward edges. The trick now is to show that this other graph of backward edges also corresponds to C_5 . That completes the proof. In general, whenever the nonconstructive proof is given, where the m -sequence is obtained at the very beginning and then some Pigeonhole Principle approach is used to get a linear set exactly adjacent to/from a big transitive chunk, our

algorithmic framework may be used. Since we do not use the regularity lemma, we do not rely on the bounds provided by this tool to obtain lower bounds on the sizes on the elements of the m -sequence. That leads to the explicit lower bounds on the EH coefficients.

Let us also introduce few small tournaments that are not constellations but play important role in the research on the conjecture for small forbidden patterns. We have already introduced C_5 - a unique tournament on 5 vertices for which every vertex has indegree 2. Let T_6 be a tournament with $V(T_6) = \{1, 2, \dots, 6\}$ such that under ordering $(1, 2, \dots, 6)$ of its vertices the only backward edges are: $(4, 1), (6, 3), (6, 1), (5, 2)$. Let T_6^1 be a tournament with $V(T_6^1) = \{1, 2, \dots, 6\}$ such that under ordering $(1, 2, \dots, 6)$ of its vertices the only backward edges are: $(4, 1), (5, 1), (5, 2), (6, 3)$. Let T_6^2 be a tournament with $V(T_6^1) = \{1, 2, \dots, 6\}$ such that under ordering $(1, 2, \dots, 6)$ of its vertices the only backward edges are: $(1, 3), (2, 3), (2, 4), (6, 5)$.

We just note that, as in the proof presented in [5], the algorithm for C_5 uses two orderings of the vertices of C_5 : ordering $(1, 2, 3, 4, 5)$, under which the set of backward edges is of the form $\{(4, 1), (5, 2), (5, 1)\}$ (the so-called *tree-ordering* since the graph of backward edges is a tree) and ordering $(4, 1, 3, 5, 2)$ (the so-called *cyclic ordering*). Surprisingly, a very similar method may be used for tournament T_6^1 and tournament T_6^2 . Note that both tournaments are prime. For tournament T_6^1 the two crucial orderings of vertices are: $(1, 2, 3, \dots, 6)$, under which the set of backward edges is of the form $\{(4, 1), (5, 1), (5, 2), (6, 3)\}$ (so-called *forest ordering*) and ordering $(2, 4, 1, 6, 3, 5)$, under which the set of backward edges is of the form $\{(5, 1), (1, 2), (5, 2), (3, 4)\}$. For a tournament T_6^2 the two crucial orderings of vertices are: $(1, 2, 3, \dots, 6)$, under which the set of backward edges is of the form $\{(4, 1), (5, 2), (5, 1), (6, 3)\}$ (so-called *forest ordering*) and ordering $(5, 2, 4, 1, 6, 3)$, under which the set of backward edges is of the form $\{(3, 4), (4, 5), (3, 5), (1, 2)\}$. Thus if $\mathcal{F} = \mathcal{C} \cup \{C_5, T_6^1, T_6^2\}$ and $\hat{\mathcal{F}}$ is the closure of \mathcal{F} under substitutions, then there exists a sub-exponential algorithm that finds a polynomial- size transitive subtournament of a H -free n -vertex tournament, where $H \in \hat{\mathcal{F}}$. Besides there exists a sub-exponential algorithm that colors any n -vertex H -free tournament with $O(n^{1-\epsilon} \log(n))$ colors, where $H \in \hat{\mathcal{F}}$.

Thus, using Theorem 6.1, we conclude that there exists a sub-exponential algorithm that colors every n -vertex H -free tournament with $O(n^{1-\epsilon} \log(n))$ colors, where H is an arbitrary tournament on at most 5 vertices or a tournament on 6 vertices different than T_6 . For those tournaments H finding polynomial-size transitive subtournaments of H -free tournaments can be also done in sub-exponential time.

It can be proven ([8]) that:

6.1 *Every tournament on at most 5 vertices is either isomorphic to C_5 or is of the form $H(F_1, \dots, F_h)$ for some constellation H with $V(H) = \{v_1, \dots, v_h\}$, $h > 1$, and some constellations F_1, \dots, F_h . Every tournament on at most 6 vertices is either isomorphic to T_6 , T_6^1 , or T_6^2 or is of the form $H(F_1, \dots, F_h)$ for some tournament H with $V(H) = \{v_1, \dots, v_h\}$, $h > 1$ and some tournaments F_1, \dots, F_h .*

The proof uses a brute-force method thus we skip it.

Thus by using our techniques one can obtain a constructive proof of the Erdős-Hajnal conjecture for all tournaments on at most 5 vertices and all tournaments on 6 vertices but T_6 . This result is interesting since in the undirected case there are still graphs on 5 vertices for which the conjecture is open. Furthermore, the conjecture is still open for all undirected graphs on 6 vertices that cannot be constructed from smaller graphs by the substitution procedure.

References

- [1] N. Alon, J. Pach, J. Solymosi, Ramsey-type theorems with forbidden subgraphs, *Combinatorica*, 155-170 (2001).
- [2] P. Erdős, A. Hajnal, Ramsey-type theorems, *Discrete Applied Mathematics*, 37-52 (1989).
- [3] K. Choromanski, Upper bounds for Erdős-Hajnal coefficients of tournaments, *JGT* (2012).
- [4] K. Choromanski, EH-suprema of tournaments with no nontrivial homogeneous sets, *submitted for publication*.
- [5] E. Berger, K. Choromanski, M. Chudnovsky, Forcing Large Transitive Subtournaments, *JCTB* (2014).
- [6] E. Berger, K. Choromanski, M. Chudnovsky, J. Fox, M. Loebl, A. Scott, P. Seymour, S. Thomassé, Tournaments and coloring, *JCTB* (2012).
- [7] K. Choromanski, M. Chudnovsky, P. Seymour, Tournaments with near-linear transitive subsets, *JCTB* (2014).
- [8] Krzysztof Choromanski, Maria Chudnovsky, *private conversation*, (2012).